Maturi Venkata Subba Rao Engineering College

(An Autonomous Institution)

Nadergul (P.O.), Hyderabad - 501 510.

(Affiliated to Osmania University, Hyderabad)
Accredited by NAAC,ISO 9001:2015 Certified

Department of Information Technology

Manual and Observation Book

CN&NS Lab
(Code: U21PE782IT)

B.E (IT) IV Year, VII Semester

Academic Year 2024-2025

Maturi Venkata Subba Rao Engineering College

(An Autonomous Institution)

Nadergul (P.O.), Hyderabad - 501 510.

(Affiliated to Osmania University, Hyderabad)
Accredited by NAAC,ISO 9001:2015 Certified

Department of Information Technology

Manual and Observation Book

CN&NS Lab
(Code: U21PE782IT)

(IT) IV Year, VII Semester

NAME ______________________________

CLASS : ______________ SECTION ______________

ROLL NO. ______________________________

Academic Year 2024-2025

U21PE782 IT

CN&NS LAB

Instruction	3 Periodsper week
Duration	3Hours
University Examination	50 Marks
Sessional	25 Marks

Course Objectives:

The student will be able

- To become familiar with basic networking commands.
- To learn how to use socket system calls
- To implement network routing algorithms.
- To use different cipher techniques
- To build authentication algorithms

CN&NS Lab:

1. Familiarization of Network Environment, Understanding and using network utilities: ipconig, ifconfig, netstat, ping, arp, telnet,ftp,finger,traceroute, whois.
2. Implementation of bit stuffing
3. Implementation of character stuffing
4. Implementation of concurrent server service using connection oriented socket system Calls (Service: Daytime, Time)
5. Implementation of concurrent server using connection less socket system calls. (Service: Echo server,String Concateation)
6. Implementation of Iterative server using connection oriented socket system calls. (Service: Calculate Employee Salary)
7. Implementation of Iterative server using connection less socket system calls. (Service: Student Grade)
8. Implementation of Distance Vector Routing Protocol.
9. Write a program to perform encryption and decryption using the following algorithms
   a. Ceaser cipher b. Substitution cipher
10. Write a program to implement the DES algorithm.
11. Write a program to implement RSA algorithm.
12. Implement Diffie-Hellman Key Exchange Algorithm.
13. Calculate the message digest of a text using the SHA-1 algorithm
14. Calculate the message digest of a text using the MD5 algorithm.
15. Case study on any open source network simulation tool.(simple routing protocol Implementation)

Text Books:

1. W. Richard Stevens, “Unix Network Programming”, Prentice Hall, Pearson Education,2009.
2. Cryptography and Network Security – Principles and Practice: William Stallings, Pearson Education 6th Edition

CN&NS Lab

CODE: U21PE782IT B.E (IT) 4/4- VII SEMESTER

COURSE OBJECTIVES AND OUTCOMES:

Course Objectives:

The student will be able

- To become familiar with basic networking commands.
- To learn how to use socket system calls
- To implement network routing algorithms.
- To use different cipher techniques
- To build authentication algorithms

Course Outcomes:

After completing this course the student will be able to:

- Understand basic networking commands
- Develop Client-Server Socket based programs using TCP and UDP sockets
- Implement network routing algorithms
- Build cryptosystems by applying symmetric and public key encryption algorithms.
- Construct code for authentication algorithms.

Department of Information Technology

CN&NS Laboratory

NAME____

CLASS :______________________________SECTION ______________________

ROLL NO. ____

INDEX

S.No.	Name of the Experiment	Date of Expt. Conducted	Date of Submission	Grade/Marks	Signature	Remarks

Signature of the Head of the Department

Signature of the Class In-charge

LIST OF EXPERIMENTS

CN Lab Programs:

1. Familiarization of Network Environment, Understanding and using network utilities: ipconig, ifconfig, netstat, ping, arp, telnet,ftp,finger,traceroute, whois.
2. Implementation of bit stuffing
3. Implementation of character stuffing
4. Implementation of concurrent server service using connection oriented socket system Calls (Service: Daytime, Time)
5. Implementation of concurrent server using connection less socket system calls. (Service: Echo server,String Concateation)
6. Implementation of Iterative server using connection oriented socket system calls. (Service: Calculate Employee Salary)
7. Implementation of Iterative server using connection less socket system calls. (Service: Student Grade)
8. Implementation of Distance Vector Routing Protocol.

NS Lab programs:

9. Write a program to perform encryption and decryption using the following algorithms
   a. Ceaser cipher b. Substitution cipher
10. Write a program to implement the DES algorithm.
11. Write a program to implement RSA algorithm.
12. Implement Diffie-Hellman Key Exchange Algorithm.
13. Calculate the message digest of a text using the SHA-1 algorithm
14. Calculate the message digest of a text using the MD5 algorithm.
15. Case study on any open source network simulation tool.(simple routing protocol Implementation)

Suggested Reading:

1. W. Richard Stevens, “Unix Network Programming”, Prentice Hall, Pearson Education, 2009.
2. Cryptography and Network Security – Principles and Practice: William Stallings, Pearson Education 6th Edition

.

LAB SCHEDULE

CN/NS LAB

Class: 4/4 Section: A, B, C Batch: A &B

Week	**Experiments**	**Date Planned**	**Date Conducted**
Week 1	Basics of UNIX commands		
Week2	Implementation of bit stuffing and character stuffing		
Week 3	Implementation of concurrent server service using connectionoriented socket system Calls (Service: Daytime, Time)		
Week 4	Implementation of concurrent server using connection less socket system calls. (Service: Echo server,String Concateation)		
Week 5	Implementation of Iterative server using connection oriented socket system calls. (Service: Calculate Employee Salary)		
Week 6	Implementation of Iterative server using connection less socket system calls. (Service: Student Grade)		
Week 7	Implementation Distance Vector Routing Protocol.		
Week 8	Write a program to perform encryption and decryption using the following algorithms : a. Ceaser cipher b. Substitution cipher		
Week 9	Write a program to implement the DES algorithm		
Week10	Write a program to Implement RSA algorithm and Diffie-Hellman Key Exchange Algorithm.		
Week11	Calculate the message digest of a text using the SHA-1 algorithm		
Week12	Calculate the message digest of a text using the MD5 algorithm		
Week13	Case study on any open source network simulation tool. (simple routing protocol Implementation)		

Note: Regular Evaluation – Procedure (15M), Execution (20M), Viva (15M) Faculty will be noted evaluated marks in Student Report and in Register

INTRODUCTION TO COMPUTER NETWORK PROGRAMMING LABORATORY

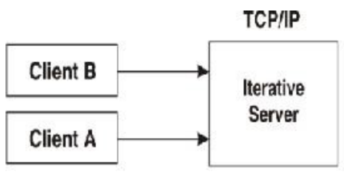
NETWORKING BASICS

Computer networking is the engineering discipline concerned with communication between computer systems or devices.
It is the practice of linking computing devices together with hardware and software that supports data communications across these devices.

KEY CONCEPTS AND TERMS

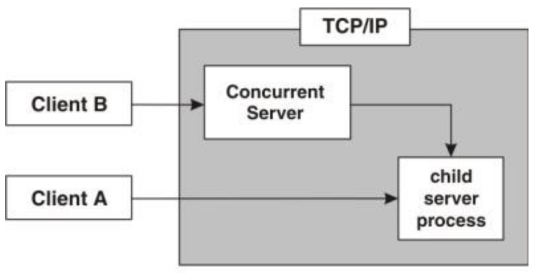
Packet A message or data unit that is transmitted between communicating processes.
Host: A computer system that is accessed by a user working at a remote location. It is the remote process with which a process communicates. It may also be referred as Peer.
Channel: Communication path created by establishing a connection between endpoints.
Network A group of two or more computer systems linked together
Server: In computer networking, a server is a computer designed to process requests and deliver data to other computers over a local network or the Internet.

- **Iterative servers**: This server knows ahead of time about how long it takes to handle eachrequest & server process handles each request itself.



Iterative Server

- **Concurrent servers:** The amount of work required to handle a request is unknown, so the server starts another process to handle each request.



Concurrent Server

Client: A Client is an application that runs on a personal computer or workstation and relies on a server to perform some operations.

Network Address: Network addresses give computers unique identities they can use to communicate with each other. Specifically, IP addresses and MAC addresses are used on most home and business networks.

Protocols: A Protocol is a convention or standard rules that enables and controls the connection, communication and data transfer between two computing endpoints.

Port An interface on a computer to which you can connect a device. It is a "logical connection place" and specifically, using the Internet's protocol, TCP/IP.

A port is a 16-bit number, used by the host-to-host protocol to identify to which higher-level protocol or application program (process) it must deliver incoming messages.

PORTS	**RANGE**
Well-known ports	1-1023
Ephemeral ports	1024-5000
User-defined ports	5001-65535

Table 1: Ports and Ranges

Connection: It defines the communication link between two processes.

Association: Association is used for 5 tuple that completely specifies the two processes that make up a connection.

{ Protocol, local-address, local-process, foreign-address, foreign- process}

The local address and foreign address specify the network ID & Host-ID of the local host and the foreign host in whatever format is specified by protocol suite.

The local process and foreign process are used to identify the specific processes on each system that are involved in a connection.

We also define Half association as either

{ protocol, local-address, local process} or ***{ protocol, local-address, local process}***

which specify each half of a connection. This half association is called a Socket or transport address.

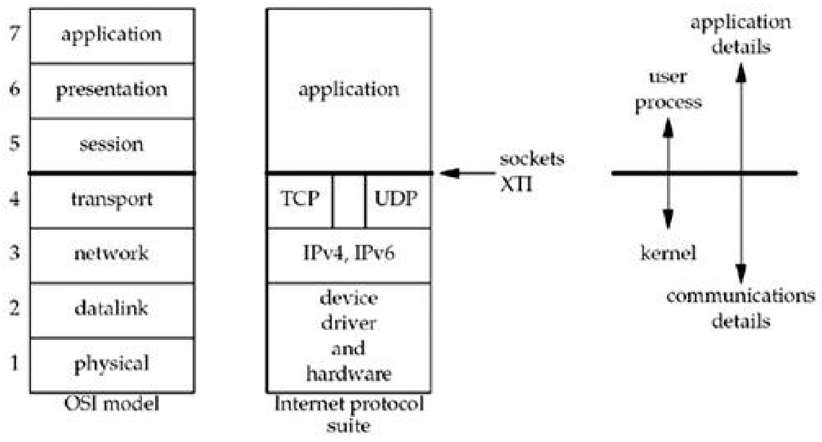
	Protocol	Local Address , Local Process	Foreign Address , Foreign Process
connection-oriented server	socket()	bind()	listen() accept()
connection-oriented client	socket()	connect()	
connectionless server	socket()	bind()	recvfrom()
connectionless client	socket()	bind()	sendto()

Connection-oriented and Connectionless Flow

OSI Model

A common way to describe the layers in a network is to use the International Organization for Standardization (ISO) open systems interconnection (OSI) model for computer communications. This is a seven-layer model, which we show in Figure below along with the approximate mapping to the Internet protocol suite.

We consider the bottom two layers of the OSI model as the device driver and networking hardware that are supplied with the system. The network layer is handled by the IPv4 and IPv6 protocols. The transport layers that we can choose from are TCP and UDP



Layers in OSI model and Internet protocol suite

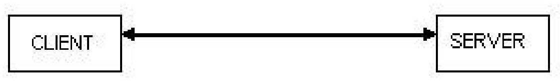
The upper three layers of the OSI model are combined into a single layer called the **application**. This is the Web client (browser) or whatever application we are using. With the Internet protocols, there is rarely any distinction between the upper three layers of the OSI model.

The sockets programming interfaces are interfaces from the upper three layers (the "application") into the transport layer. The sockets provide the interface from the upper three layers of the OSI model into the transport layer. There are two reasons for this design:

- The upper three layers handle all the details of the application and know little about the communication details. The lower four layers know little about the application, but handle all the communication details: sending data, waiting for acknowledgments, and so on.
- The second reason is that the upper three layers often form what is called a user process while the lower four layers are normally provided as part of the operating system (OS) kernel.

CLIENT-SERVER MODEL

Network applications can be divided into two processes: a Client and a Server, with a communication link joining the two processes.



Client Server Model

Normally, from Client-side it is one-one connection. From the Server Side, it is many-one connection.
The standard model for network applications is the Client-Server model. A Server is a process that is waiting to be contacted by a Client process so that server can do something for the client.
Typical BSD Sockets applications consist of two separate application level processes; one process (the **client**) requests a connection and the other process (the **server**) accepts it.

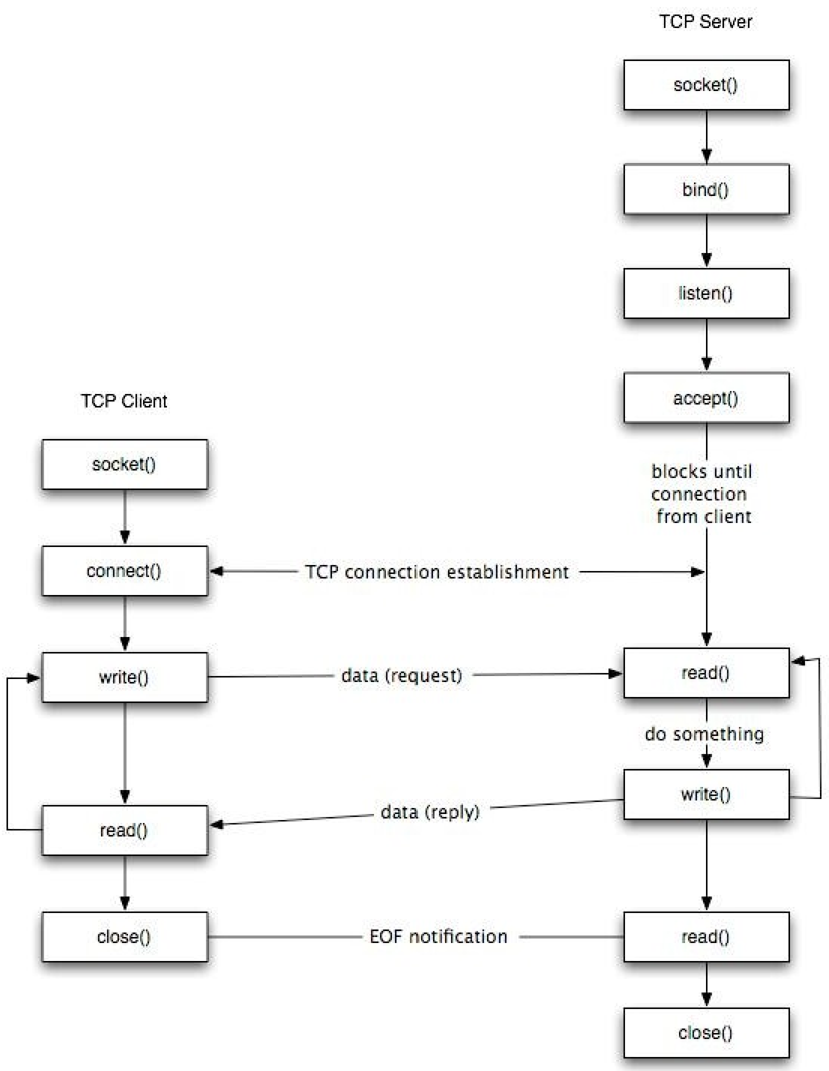


Fig 2.6: Socket functions for elementary TCP client/server in Connection-oriented Scenario

The server process creates a socket, binds an address to it, and sets up a mechanism (called a listen queue) for receiving connection requests. The client process creates a socket and requests a connection to the server process. Once the server process accepts a client process's request and establishes a connection, full-duplex (two-way) communication can occur between the two sockets.

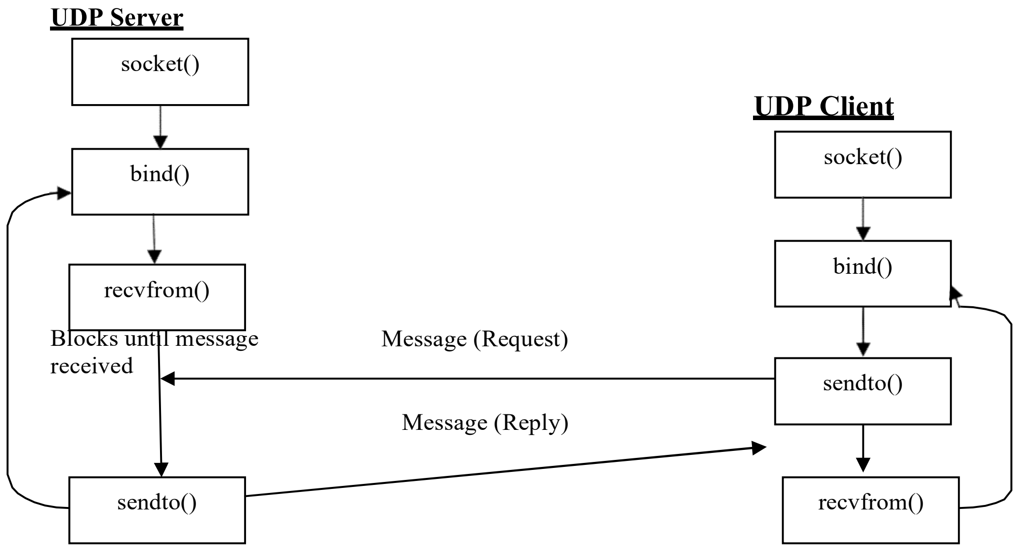
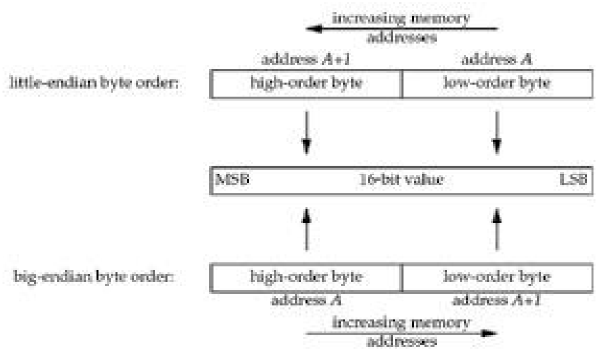


Fig 2.7: Diagram showing the sequence of steps to be followed by the Connectionless Iterative server and client

The server process creates a socket, binds an address to it, and waits for receiving request messages from client. The client process creates a socket and binds an address to it,the server process accepts a client process's request message and sends the reply message, full-duplex (two-way) communication can occur between the two sockets.

Byte-Ordering Functions: Consider a 16-bit integer that is made up of 2 bytes. There are two ways to store the two bytes in memory: with the low-order byte at the starting address, known as little-endian byte order, or with the high-order byte at the starting address, known as big-endian byte order. Little-endian byte order and big-endian byte order for a 16-bit integer.



Big -endian and Little-endian Byte order

In this figure, we show increasing memory addresses going from right to left in the top, and from left to right in the bottom. We also show the most significant bit (MSB) as the leftmost bit of the 16-bit value and the least significant bit (LSB) as the rightmost bit. The terms "little-endian" and "big-endian" indicate which end of the multibyte value, the little end or the big end, is stored at the starting address of the value.

We refer to the byte ordering used by a given system as the *host byte order*. We must deal with these byte ordering differences as network programmers because networking protocols must specify a *network byte order*. Our concern is therefore converting between host byte order and network byte order. We use the following four functions to convert between these two byte orders.

```
#include <netinet/in.h>
#include <sys/types.h>
unsigned long htonl(unsigned long hostlong);
unsigned short htons(unsigned short hostshort);
unsigned long ntohl(unsigned long netlong);
unsigned short ntohs(unsigned short netshort);
```

Htons	host to network short
Htonl	host to network long
Ntohs	network to host short
Ntohl	network to host long

Sockets Overview

The operating system includes the Berkeley Software Distribution (BSD) interprocess communication (IPC) facility known as *sockets*. Sockets are communication channels that enable unrelated processes to exchange data locally and across networks. A single socket is one end point of a two-way communication channel.

In the operating system, sockets have the following characteristics:

- A socket exists only as long as a process holds a descriptor referring to it.
- Sockets are referenced by file descriptors and have qualities similar to those of a character special device. Read, write, and select operations can be performed on sockets by using the appropriate subroutines.
- Sockets can be created in pairs, given names, or used to rendezvous with other sockets in a communication domain, accepting connections from these sockets or exchanging messages with them.

Sockets Background: Sockets were developed in response to the need for sophisticated interprocess facilities to meet the following goals:

- Provide access to communications networks such as the Internet.

- Enable communication between unrelated processes residing locally on a single host computer and residing remotely on multiple host machines.

Socket Facilities: Socket subroutines and network library subroutines provide the building blocks for IPC. An application program must perform the following basic functions to conduct IPC through the socket layer:

- Create and name sockets.
- Accept and make socket connections.
- Send and receive data.
- Shut down socket operations.

Socket Interface: The Socket interface provides a standard, well-documented approach to access kernel network resources.

Socket Header Files to be Included: Socket header files contain data definitions, structures, constants, macros, and options used by socket subroutines. An application program must include the appropriate header file to make use of structures or other information a particular socket subroutine requires. Commonly used socket header files are:

/usr/include/netinet/in.h	Defines Internet constants and structures.
/usr/include/netdb.h	Contains data definitions for socket subroutines.
/usr/include/sys/socket.h	Contains data definitions and socket structures.
/usr/include/sys/types.h	Contains data type definitions.
/usr/include/arpa.h	Contains definitions for internet operations.
/usr/include/sys/errno.h	Defines the **errno**values that are returned by drivers and other kernel-level code.

Internet address translation subroutines require the inclusion of the **inet.h**file. The **inet.h**file is located in the **/usr/include/arpa**directory.

Socket Addresses: Sockets can be named with an address so that processes can connect to them. Most socket functions require a pointer to a socket address structure as an argument. Each supported protocol suite defines its own socket address structure. The names of these structures begin with sockaddr_ and end with a unique suffix for each protocol suite.

Generic socket address structure: Many of the Networking system calls require a pointer to a socket address structure as an argument.

Definition of this structure is in

```
structsockaddr {
                unsigned short sa_family; /* address family : AF_xxx Value */
                        charsa_data[14]; /* up to 14 bytes of protocol- specific address
                        */
```

Internet Socket address structure: The protocol specific structure sockaddr_in is identical in size to generic structure which is 16 bytes.

```
#include <netinet/in.h>
structsockaddr_in {
                    shortsin_family; /* AF_INET
                    unsigned short sin_port; /* 16-bit port number */

                    /* Network-byte ordered */
                    structin_addrsin_addr; /* 32-bit netid/hostid*/

                    /* Network-byte ordered */
                    charsin_zero[8]; /* unused*/
             };

structin_addr {
                    unsigned long s_addr; /* 32-bit netid/hostid */
              /* network byte ordered*/     };
```

sin_zerois unused member, but we always set it to 0 when filling in one of these structures. Socket address structures are used only on a given host: the structure itself is now communicated between different hosts, although certain fields (eg: IP Address & ports) are used for communication. *The protocol-specific structure **sockaddr_in**is identical in **size** to generic structure sockaddr which is **16 bytes**

<u>Basic Unix commands</u>

Command	CAT
Syntax	cat [argument] [specific file]
Description	“cat" is short for concatenate. This command is used to create, view and concatenate files.
Examples	cat /etc/passwd This command displays the "/etc/passwd" file on your screen. cat /etc/profile This command displays the "/etc/profile" file on your screen. Notice that some of the contents of this file may scroll off of your screen. cat file1 file2 file3 > file4 This command combines the contents of the first three files into the fourth file.
Command	pwd
Syntax	pwd
Description	"pwd" stands for parent working directory. It displays your current position in the UNIX filesystem.
Examples	pwd There are no options (or arguments) with the "pwd" command. It is simply used to report your current working directory.
Command	Ls
Syntax	ames]
Description	"ls" stands for list. It is used to list information about files and directories.

Examples	ls This is the basic "ls" command, with no options. It provides a very basic listing of the files in your current working directory. Filenames beginning with a decimal are considered *hidden* files, and they are not shown. ls -a The -a option tells the ls command to report information about all files, including hidden files.
	ls -l The -l option tells the "ls" command to provide a *long* listing of information about the files and directories it reports. The long listing will provide important information about file permissions, user and group ownership, file size, and creation date. ls -al This command provides a *long* listing of information about *all* files in the current directory. It combines the functionality of the -a and -l options. *This is probably the most used version of the ls command.* ls -al /usr This command lists long information about all files in the "/usr" directory. ls -alR /usr | more This command lists long information about all files in the "/usr" directory, and all sub-directories of /usr. The -R option tells the ls command to provide a *recursive* listing of all files and sub-directories. ls -ld /usr Rather than list the files contained in the /usr directory, this command lists information about the /usr directory itself (without generating a listing of the contents of /usr). This is very useful when you want to check the permissions of the directory, and not the files the directory contains.
Command	Mv
Syntax	mv [ptions] sources target
Options	-b backup files that are about to be overwritten or removed -i interactive mode; if dest exists, you'll be asked whether to overwrite the file
Description	The "mv" command is used to move and rename files.
Examples	mv Chapter1 Chapter1.bad This command renames the file "Chapter1" to the new name "Chapter1.bad". mv Chapter1 garbage This command renames the file "Chapter1" to the new name "garbage". (Notice that if "garbage" is a directory, "Chapter1" would be moved into that directory). mv Chapter1 /tmp This command moves the file "Chapter1" into the directory named "/tmp". mv tmp tmp.old Assuming in this case that tmp is a directory, this example renames the directory tmp to the new name tmp.old.
Command	Rm
Syntax	Iles

Options	-d, --directory unlink FILE, even if it is a non-empty directory (super-user only) -f, --force
	ignore nonexistent files, never prompt -i, --interactive prompt before any removal -r, -R, --recursive remove the contents of directories recursively -v, --verbose explain what is being done
Description	The "rm" command is used to remove files and directories. (Warning - be very careful when removing files and directories!)
Examples	rm Chapter1.bad This command deletes the file named "Chapter1.bad" (assuming you have permission to delete this file). rm Chapter1 Chapter2 Chapter3 This command deletes the files named "Chapter1", "Chapter2", and "Chapter3". rm -i Chapter1 Chapter2 Chapter3 This command prompts you before deleting any of the three files specified. The -i option stands for *inquire*. You must answer y (for yes) for each file you really want to delete. This can be a safer way to delete files. rm *.html This command deletes all files in the current directory whose filename ends with the characters ".html". rm index* This command deletes all files in the current directory whose filename begins with the characters "index". rm -r new-novel This command deletes the directory named "new-novel". This directory, and all of its' contents, are erased from the disk, including any sub-directories and files.
Command	Cp
Syntax	ile1 file2 iles directory
Options	-b backup files that are about to be overwritten or removed -i interactive mode; if dest exists, you'll be asked whether to overwrite the file -p preserves the original file's ownership, group, permissions, and timestamp
Description	The "cp" command is used to copy files and directories. Note that when using the cp command, you must always specify both the source and destination of the file(s) to be copied.
Examples	cp .profile .profile.bak This command copies your ".profile" to a file named ".profile.bak". cp /usr/fred/Chapter1 . This command copies the file named "Chapter1" in the "/usr/fred" directory to the current directory. This example assumes that you have write permission in the current directory. cp /usr/fred/Chapter1 /usr/mary

	This command copies the "Chapter1" file in "/usr/fred" to the directory named "/usr/mary". This example assumes that you have write permission in the "/usr/mary" directory.
Command	Grep
Syntax	] regular expression [files]
Options	-i case-insensitive search -n show the line# along with the matched line -v invert match, e.g. find all lines that do NOT match -w match entire words, rather than substrings
Description	Think of the "grep" command as a "search" command (most people wish it was named "search"). It is used to search for text strings within one or more files.
Examples	grep 'fred' /etc/passwd This command searches for all occurrences of the text string 'fred' within the "/etc/passwd" file. It will find and print (on the screen) all of the lines in this file that contain the text string 'fred', including lines that contain usernames like "fred" - and also "alfred". grep '^fred' /etc/passwd This command searches for all occurrences of the text string 'fred' within the "/etc/passwd" file, but also requires that the "f" in the name "fred" be in the first column of each record (that's what the caret character tells grep). Using this more-advanced search, a user named "alfred" would not be matched, because the letter "a" will be in the first column. grep 'joe' * This command searches for all occurrences of the text string 'joe' within all files of the current directory.
Command	Mkdir
Syntax	mkdir [options] directory name
Description	The "mkdir" command is used to create new directories (sub-directories).
Examples	mkdir tmp This command creates a new directory named "tmp" in your current directory. (This example assumes that you have the proper permissions to create a new sub-directory in your current working directory.) mkdir memos letters e-mail This command creates three new sub-directories (memos, letters, and e-mail) in the current directory. mkdir /usr/fred/tmp This command creates a new directory named "tmp" in the directory "/usr/fred". "tmp" is now a sub-directory of "/usr/fred". (This example assumes that you have the proper permissions to create a new directory in /usr/fred.) mkdir -p /home/joe/customer/acme This command creates a new directory named /home/joe/customer/acme, and creates any intermediate directories that are needed. If only /home/joe existed to begin with, then the
	directory "customer" is created, and the directory "acme" is created inside of customer.
Command	Rmdir

Syntax	rmdir [options] directories
Description	The "rm" command is used to remove files and directories. (Warning - be very careful when removing files and directories!)
Examples	rm Chapter1.bad This command deletes the file named "Chapter1.bad" (assuming you have permission to delete this file). rm Chapter1 Chapter2 Chapter3 This command deletes the files named "Chapter1", "Chapter2", and "Chapter3". rm -i Chapter1 Chapter2 Chapter3 This command prompts you before deleting any of the three files specified. The -i option stands for *inquire*. You must answer y (for yes) for each file you really want to delete. This can be a safer way to delete files. rm *.html This command deletes all files in the current directory whose filename ends with the characters ".html". rm index* This command deletes all files in the current directory whose filename begins with the characters "index". rm -r new-novel This command deletes the directory named "new-novel". This directory, and all of its' contents, are erased from the disk, including any sub-directories and files.
Command	cd, chdir
Syntax	lirectory you want to move to]
Description	"cd" stands for change directory. It is the primary command for moving around the filesystem.
Examples	cd /usr This command moves you to the "/usr" directory. "/usr" becomes your current working directory. cd /usr/fred Moves you to the "/usr/fred" directory. cd /u*/f* Moves you to the "/usr/fred" directory - if this is the only directory matching this wildcard pattern. cd Issuing the "cd" command without any arguments moves you to your *home* directory. cd - Using the Korn shell, this command moves you back to your previous working directory. This is very useful when you're in the middle of a project, and keep moving back-and-forth between two directories.
Command	Kill
Syntax	Ds
Description	kill ends one or more process IDs. In order to do this you must own the process or be designated a privileged user. To find the process ID of a certain job use ps.
Examples	
Command	Ps
Syntax	

Description	The "ps" command (process statistics) lets you check the status of processes that are running on your Unix system.
Examples	ps The ps command by itself shows minimal information about the processes *you* are running. Without any arguments, this command will not show information about otherprocessesrunningonthesystem. ps -f The -f argument tells ps to supply *full* information about the processes it displays. In this example, ps displays full information about the processes *you* are running. ps -e The -e argument tells the ps command to show *every* process running on the system. ps -ef The -e and -f arguments are normally combined like this to show full information about every process running on the system. *This is probably the most often-used form of the ps command.* ps -ef | more Because the output normally scrolls off the screen, the output of the ps -ef command is often piped into the more command. The more command lets you view one screenful of information at a time. ps -fu fred This command shows full information about the processes currently being run by the user named *fred* (the -u option lets you specify a username).

PROCEDURETORUNC PROGRAMS:

Step 1 : Use an editor, such as vi, ex, or ed to write the program. The name of the file containing the program should end in .c.

For example, the file show.c contains the following lines: main()

```
{
      printf(" welcome to MVSR ");
}
```

Step 2 : Submit the file to CC (the C Compiler)

```
$ cc show.c
```

If the program is okay, the compiled version is placed in a file called a.out Step 3 :

To run the program, type a.out

```
$ a.out
```

CN&NS PROGRAMS

PROGRAM No. 1

AIM: Understanding and using the following network utility commands like ifconfig, netstat, ping, arp, telnet, ftp, finger, traceroute, whois.

AIM: Understanding and using of commands like ifconfig, netstat, ping, arp, telnet, ftp, finger, traceroute, whois.

INTRODUCTION:

UNIX utilities are commands that, generally, perform a single task. It may be as simple as printing the date and time, or a complex as finding files that match many criteria throughout a directory hierarchy

IFCONFIG

The UNIX command **ifconfig**(short for **i**nter**f**ace **config**urator) serves to configure and control TCP/IP network interfaces from a command line interface (CLI).
Common uses for ifconfig include setting an interface's IP address and netmask, and disabling or enabling a given interface.

NETSTAT

netstat(**net**work **stat**istics) is a command-line tool that displays network connections (both incoming and outgoing), routing tables, and a number of network interface statistics.
It is used for finding problems in the network and to determine the amount of traffic on the network as a performance measurement.

Parameters

Parameters used with this command must be prefixed with a hyphen (-) rather than a slash (/).

-a: Displays **a**ll active TCP connections and the TCP and UDP ports on which the computer is listening.

-e: Displays **e**thernet statistics, such as the number of bytes and packets sent and received. This parameter can be combined with **-s**.

-f: Displays **f**ully qualified domain names <FQDN> for foreign addresses.

-i: Displays network **i**nterfaces and their statistics (not available under Windows)

-n: Displays active TCP connections, however, addresses and port numbers are expressed numerically and no attempt is made to determine names.

-o: Displays active TCP connections and includes the process ID (PID) for each connection.

-p Linux: **P**rocess: Show which processes are using which sockets

PING

Ping is a computer network tool used to test whether a particular host is reachable across an IP network; it is also used to self-test the network interface card of the computer, or as a speed test. It works by sending ICMP “echo request” packets to the target host and listening for ICMP “echo response" replies. Ping does not estimate the round-trip time, as it does not factor in the user's connection speed, but instead is used to record any packet loss, and print a statistical summary when finished.

ARP

In computer networking, the **Address Resolution Protocol** (**ARP**) is the method for finding a host's link layer (hardware) address when only its Internet Layer (IP) or some other Network Layer address is known.

ARP has been implemented in many types of networks; it is not an IP-only or Ethernet-only protocol. It can be used to resolve many different network layer protocol addresses to interface hardware addresses, although, due to the overwhelming prevalence of IPv4 and Ethernet, ARP is primarily used to translate IP addresses to Ethernet MAC addresses.

TELNET

Telnet (**Tel**ecommunication **net**work) is a network protocol used on the Internet or local area network (LAN) connections.Typically, telnet provides access to a command-line interface on a remote machine.The term *telnet* also refers to software which implements the client part of the protocol. Telnet clients are available for virtually all platforms.

Protocol details:

Telnet is a client-server protocol, based on a reliable connection-oriented transport. Typically this protocol is used to establish a connection to TCP port 23

FTP File Transfer Protocol (**FTP**):

FTP is a network protocol used to transfer data from one computer to another through a network such as the Internet .FTP is a file transfer protocol for exchanging and manipulating files over a TCP computer network. An FTP client may connect to an FTP server to manipulate files on that server. FTP runs over TCP. It defaults to listen on port 21 for incoming connections from FTP clients. A connection to this port from the FTP Client forms the control stream on which commands are passed from the FTP client to the FTP server and on occasion from the FTP server to the FTP client. FTP uses out-of-band control, which means it uses a separate connection for control and data. Thus, for the actual file transfer to take place, a different connection is required which is called the data stream.

FINGER

In computer networking, the **Name/Finger protocol** and the **Finger user information protocol** are simple network protocols for the exchange of human-oriented status and user information.

TRACEROUTE

Traceroute is a computer network tool used to determine the route taken by packets across an IP network. An IPv6 variant, **traceroute6**, is also widely available. Traceroute is often used for network troubleshooting. By showing a list of routers traversed, it allows the user to identify the path taken to reach a particular destination on the network. This can help identify routing problems or firewalls that may be blocking access to a site. Traceroute is also used by penetration testers to gather information about network infrastructure and IP ranges around a given host. It can also be used when downloading data, and if there are multiple mirrors available for the same piece of data, one can trace each mirror to get a good idea of which mirror would be the fastest to use.

WHO IS:

WHOIS (pronounced "**who is**"; not an acronym) is a query/response protocol which is widely used for querying an official database in order to determine the owner of a domain name, an IP address, or an autonomous system number on the Internet. WHOIS lookups were traditionally made using a command line interface, but a number of simplified web-based tools now exist for looking up domain ownership details from different databases. WHOIS normally runs on TCP port 43.

The WHOIS system originated as a method that system administrators could use to look up information to contact other IP address or domain name administrators (almost like a "white pages").

PROGRAM No. 02

Aim: Implementation of bit stuffing :

```
//Bit stuffing is a process of inserting an extra bit as 0, once the frame sequence encountered 5 consecutive 1's.

#include<stdio.h>
#include<string.h>
int main()
{
  int a[20],b[30],i,j,k,count,n;
  printf("Enter frame size (Example: 8):");
  scanf("%d",&n);
  printf("Enter the frame in the form of 0 and 1 :");
  for(i=0; i<n; i++)
    scanf("%d",&a[i]);
  i=0;
  count=1;
  j=0;
  while(i<n)
  {
    if(a[i]==1)
    {
      b[j]=a[i];
      for(k=i+1; a[k]==1 && k<n && count<5; k++)
      {
        j++;
        b[j]=a[k];
        count++;
        if(count==5)
        {
          j++;
          b[j]=0;
        }
        i=k;
      }
    }
    else
    {
      b[j]=a[i];
    }
    i++;
    j++;
  }
  printf("After Bit Stuffing :");
  for(i=0; i<j; i++)
    printf("%d",b[i]);
  return 0;
}
```

OUTPUT for BIT STUFFING:

```
Enter frame size (Example: 8):12
Enter the frame in the form of 0 and 1 :0 1 0 1 1 1 1 1 1 0 0 1
After Bit Stuffing :0101111101001
```

PROGRAM No. 03
<u>Aim: Implementation of Character Stuffing:</u>

```
#include<stdio.h>
#include<string.h>
void main()
{
    char a[30], fs[50] = " ", t[3], sd, ed, x[3], s[3], d[3], y[3], ds[50] = " " ;
    int i,len;
    printf("Enter characters to be stuffed:");
    scanf("%s", a);
    printf("\nEnter a character that represents starting delimiter:");
    scanf(" %c", &sd);
    printf("\nEnter a character that represents ending delimiter:");
    scanf(" %c", &ed);
    x[0] = s[0] = s[1] = sd;
    x[1] = s[2] = '\0';
    y[0] = d[0] = d[1] = ed;
    d[2] = y[1] = '\0';
    strcat(fs, x);
    for(i=0; i < strlen(a); i++)
    {
      t[0] = a[i];
      t[1] = '\0';
      if(t[0] == sd)
        strcat(fs, s);
      else if(t[0] == ed)
        strcat(fs, d);
      else
        strcat(fs, t);
    }
    strcat(fs, y);
    printf("\n After stuffing:%s", fs);
  //Character De stuffing from Stuffing
  len=strlen(fs);
   for(i=2;i<len-1;i++)
      {
           fs[i-1]=fs[i];
      }
      fs[i-1]='\0';
      printf("\n After De stuffing:%s", fs);
```

```
 }
```
OUTPUT:

```
Enter characters to be stuffed:MVSR

Enter a character that represents starting delimiter:$

Enter a character that represents ending delimiter:#

 After stuffing: $MVSR#
 After De stuffing: MVSR
```

PROGRAM No. 04

Aim: Implementation of Concurrent server service using connection oriented socket system calls(Service: Daytime, Time)

INTRODUCTION: A connection-oriented server can be threaded so that it can serve multiple clients concurrently/simultaneously. Such a server is said to be a ***concurrent server.*** When the amount of time to service a request depends on the request itself, the server typically handles it in a concurrent fashion.

Algorithms:

Server

1. Create a socket in internet domain using socket system call and then bind with the server socket address by using bind system call.
2. Open a passive connection to table the connection request from client by using listen call.
3. Take the connection request from the client, establish a connection to server using accept.
4. A reply message is sent to client for the connection to be established and message to be displayed.
5. Release the connection using close system call.

Client

1. Create a socket in the internet domain using socket system call.
2. Establish connection with the server by specifying the server's address using connect.
3. After the connection is established with the server, a request message is sent to the server to process using write system call.
4. The response from the server is accepted by read system call.
5. Release the connection by using close.

Time Server:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h> #include
<time.h>
main(int argc,char *argv[])
{
  int sockfd,newsockfd,clilen,i,pid; char
   buffer[512],a[50];
   long t;
   char *st;
  struct sockaddr_in servaddr,cliaddr; if(argc
  !=3)
  {
     printf("Usage:server <portno>\n"); exit(1);
  }
  sockfd=socket(AF_INET,SOCK_STREAM,0); if(sockfd
  < 0)
  {
    printf("error server socket\n");
    exit(1);
 }
   servaddr.sin_family = AF_INET;
   servaddr.sin_addr.s_addr=inet_addr(argv[2]);
 servaddr.sin_port=htons(atoi(argv[1]));
  if(bind(sockfd,(struct sockaddr *)&servaddr,sizeof(servaddr)) < 0)
  {
    printf("error in bind");
    exit(1);
 }
  if(listen(sockfd,5) < 0)
  {
    printf("listen");
    exit(1);
  }
 for(; ;)
  {
     clilen=sizeof(cliaddr);
     newsockfd=accept(sockfd,(struct sockaddr *)&cliaddr,&clilen); if(newsockfd<0)
     {
       printf("accept");
       exit(1);
     }
     if((pid==fork())<0)
     {
        printf("server failed to creat child");
        exit(1);
     }
     else
     {
//      close(sockfd);

   while(i=read(newsockfd,buffer,sizeof(buffer))!=0)

       {
         if(i<0)
         {
         printf("error in read");
         exit(1);
         }
         t=time(&t);
       st=(char *)ctime(&t);
       strcpy(buffer,st);
       i=strlen(st);
      // read(newsockfd,a,50);
       printf("server received %s %s",a,buffer);
//      printf("\n%s%s",a,buffer);
```

```
      if(write(newsockfd,buffer,i)!=i)
      {
       printf("error in write\n"); exit(1);
    }
    } /*while*/
    //exit(0);
     } /*pid*/
close(newsockfd);
}
}
```

Output:

```
$ cc -o a tcpconserver.c
$ ./a 8050 172.0.5.48
server received   Fri Apr  6 01:12:47 2021
server received   Fri Apr  6 01:13:53 2021
```

Time Client:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
main(int argc,char *argv[])
{
  int sockfd,i;
  int read_frm_stdin,read_frm_sock;
  char buffer[512];
  struct sockaddr_in servaddr;
  if(argc !=3)
  {
     printf("Client :Usage:client <portno> <server_name>\n"); exit(1);
  }
  sockfd=socket(AF_INET,SOCK_STREAM,0); if(sockfd
  < 0)
  {
    perror("socket");
    exit(1);
 }
   servaddr.sin_family = AF_INET;
   servaddr.sin_addr.s_addr=inet_addr(argv[2]);
   servaddr.sin_port=htons(atoi(argv[1]));
   if(connect(sockfd,(struct sockaddr *)&servaddr,sizeof(servaddr)) < 0)
   {
     perror("connect");
     exit(1);
   }
 write(sockfd,"the time and date is",50);
      read(sockfd,buffer,sizeof(buffer));
     printf("Client received the time and date is:%s\n",buffer);
     close(sockfd);
}
```

Output:

For 1st Client:

```
$ cc -o aa tcptimeclient.c
$ ./aa 8050 172.0.5.48
```

```
Client received the time and date is:Fri Apr 6 01:12:47 2021
```

For 2nd Client:

```
$ cc -o bb tcptimeclient.c
$ ./bb 8050 172.0.5.48
Client received the time and date is:Fri Apr 6 01:13:53 2021
```

PROGRAM No. 5

Aim: Implementation of Concurrent server using connection less socket system calls(Service: Echo server)

INTRODUCTION: The server knows ahead of time about how long it takes to handle eachrequest and server process handles each request itself is known as iterative server.

Connection less Implementation Server

- Create a UDP Socket.
- Fill in the socket address structure (with server information)
- Specify the port where the service will be defined to be used by client.
- Bind the address and port using *bind()* system call.
- Receive a message from the Client using *recvfrom()* system call.
- Send the result of the request made by the client using *sendto()* system call.

Client

- Create a UDP Socket.
- Fill in the socket address structure (with server information)
- Specify the port of the Server, where it is providing service
- For echo server, send a message to the server to be echoed using *sendto()* system call.
- Receive the result of the request made to the server using *recvfrom()* system call.
- Write the result thus obtained on the standard output.

Connectionless Concurrent Server:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
void togglecase(char *a,int cnt); int
main(int argc,char *argv[])
{
        int  sockfd,newsockfd,clilen,n; struct
        sockaddr_in servaddr,cliaddr; char
        a[50];
         int cnt=80;
        int pid;
        sockfd=socket(AF_INET,SOCK_DGRAM,0);
        if(argc!=2)
        {
          printf("Usage: <server> <portno>");
          exit(0);
          }
        if(sockfd < 0)
        {
        printf("can't create"); exit(0);
        }
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=inet_addr("172.0.5.48");
        servaddr.sin_port=htons(atoi(argv[1]));
        if(bind(sockfd,(struct sockaddr *)&servaddr,sizeof(servaddr))<0 )
        {
                printf("can't bind");
exit(0);
        }
        while(1)
        {
              clilen=sizeof(cliaddr);
              if(recvfrom(sockfd,a,50,0,(struct sockaddr *)&cliaddr,&clilen) <0)
        {
                 printf("error in recvfrom\n"); exit(0);
              }
              printf("server recd the message:%s\n",a);
              pid=fork();
              if(pid ==0)
              {
                        //printf("server received message:%s",a);
                        togglecase(a,cnt);
              if(sendto(sockfd,a,50,0,(struct  sockaddr*)&cliaddr,sizeof(cliaddr)                          )<0)
                 {
                    printf("error in sendto\n"); exit(0);
                 }
                 }
                 }
                   close(sockfd);
        }
            void togglecase(char *a,int cnt)
            {
              int i; for(i=0;i<cnt;i++)
              {
if((a[i] >='A') && (a[i] <= 'Z'))
                    a[i] += 0X20;
                 else if((a[i] >='a') && (a[i] <='z')) a[i]
                    -= 0X20;
              }
          }
```

Output:

```
$ cc -o ab clconserver.c
$ ./ab 4050
server recd the message: hi
server recd the message:  hello
```

Connectionless Concurrent Client:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
int main(int argc,char *argv[])
{
        int sockfd,servlen; char
        a[50],a1[50];
        struct sockaddr_in servaddr,cliaddr;
        sockfd=socket(AF_INET,SOCK_DGRAM,0);
        if(argc!=3)
        {
           printf("usage:<client> <port> <ipaddress>\n");
           exit(0);
        }
        if(sockfd < 0)
        {
              printf("can't bind");
              exit(0);
        }
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=inet_addr(argv[2]);
        servaddr.sin_port=htons(atoi(argv[1]));
        cliaddr.sin_family=AF_INET;
        cliaddr.sin_addr.s_addr=inet_addr(argv[2]);
        cliaddr.sin_port=htons(0);
        bind(sockfd,(struct sockaddr*)&cliaddr,sizeof(cliaddr));
        printf("enter a mess");
        fgets(a,50,stdin);
servlen=sizeof(servaddr);
        sendto(sockfd,a,50,0,(struct sockaddr*)&servaddr,servlen);
        recvfrom(sockfd,a1,50,0,(struct sockaddr*)&servaddr,&servlen);
        printf("client received msg:%s",a1);
}
```

Output:

For 1st Client:

```
$ cc -o aa clconclient.c
$ ./aa 4050 172.0.5.48
enter a mess hi
client received msg: HI
```

For 2st Client

```
$ cc -o vv clconclient.c
$ ./vv 4050 172.0.5.48
enter a mess hello
client received msg: HELLO
```

PROGRAM No. 06

<u>Aim:</u> Implementaion of Iterative server using connection oriented socket system calls (Service: Echo Service, Calculate EmploySalary)

<u>Iterativeserver:</u>

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
main(int argc,char *argv[])
{
  int sockmain,sockcli,i,j;
  int child;
  char buffer[512];
  struct sockaddr_in servaddr,cliaddr;
  if(argc !=3)
  {
     printf("Usage:server <portno>\n");
     exit(1);
  }
  sockmain=socket(AF_INET,SOCK_STREAM,0);
  if(sockmain < 0)
  {
    printf("error in server socket");
    exit(1);
 }
   servaddr.sin_family = AF_INET;
   servaddr.sin_addr.s_addr=inet_addr(argv[2]);
   servaddr.sin_port=htons(atoi(argv[1]));

  if(bind(sockmain,(struct sockaddr *)&servaddr,sizeof(servaddr)) < 0)
  {
    printf("error in bind");
 exit(1);
  }
  if(listen(sockmain,5) < 0)
  {
    printf("error in listen");
    exit(1);
  }
// for(; ;)
 // {
     i=sizeof(cliaddr);
     sockcli=accept(sockmain,(struct sockaddr *)&cliaddr,&i);
     if(sockcli<0)
      {
       printf("error in accept");
       exit(1);
     }
  //   if((child = fork()) < 0)
    // {
     //  perror("server: Failed to create child\n");
       //exit(1);
//     }
     // if(child == 0)
     // {
       // close(sockmain);
 read(sockcli,buffer,sizeof(buffer));
      printf("server received message %s\n",buffer);
     write(sockcli,buffer,sizeof(buffer));
close(sockcli);
    }
```

Output:

```
$ cc -o aa iserver.c
$ ./aa 5040 172.0.5.48
Server received message welcome
```

Iterativeclient:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
main(int argc,char *argv[])
{
  int sockmain,sockcli,i,j;
  int child;
  char buffer[50],buffer1[50];
  struct sockaddr_in servaddr,cliaddr;
  if(argc !=3)
  {
     printf("Usage:client <portno> <server>\n");
     exit(1);
  }
  sockmain=socket(AF_INET,SOCK_STREAM,0);
  if(sockmain < 0)
  {
    printf("socket");
    exit(1);
 }
   servaddr.sin_family = AF_INET;
   servaddr.sin_addr.s_addr=inet_addr(argv[2]);
   servaddr.sin_port=htons(atoi(argv[1]));
   if(connect(sockmain,(struct sockaddr *)&servaddr,sizeof(servaddr)) < 0)
    {
 printf("error in connect\n");
 exit(0);
   }
  // for(;;)
//{
   printf("enter a message\n");
   fgets(buffer,70,stdin);
  write(sockmain,buffer,sizeof(buffer));
   read(sockmain,buffer1,sizeof(buffer1));
  printf("client recd the message %s\n",buffer1);
close(sockmain);
 }
```

Output:

```
$ cc -o bb iclient.c
$ ./bb 5040 172.0.5.48
enter a message
welcome client recd the message welcome
```

PROGRAM No. 07

Aim: Implementation of Iterative server using connection less socket system calls (Service: Student Grade)

Gradeserver:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
void togglecase(char *a,int cnt);
int main(int argc,char *argv[])
{
        int sockfd,newsockfd,clilen,n; struct
        sockaddr_in servaddr,cliaddr; char
        a[50];
         int cnt=80;
        int pid;
        sockfd=socket(AF_INET,SOCK_DGRAM,0);
        if(argc!=2)
        {
          printf("Usage: <server> <portno>");
          exit(0);
        }
        if(sockfd < 0)
        {
                printf("can't create");
                exit(0);
        }
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=inet_addr("172.0.5.48");
        servaddr.sin_port=htons(atoi(argv[1]));
        if(bind(sockfd,(struct sockaddr *)&servaddr,sizeof(servaddr))<0 )
        {
                printf("can't bind");
exit(0);
        }
        //while(1)
        //{
              clilen=sizeof(cliaddr);
              if(recvfrom(sockfd,a,50,0,(struct sockaddr *)&cliaddr,&clilen) <0)
    {
                printf("error in recvfrom\n");
                exit(0);
              }
              printf("server recd the message:%s\n",a);
           //    pid=fork();
             //  if(pid ==0)
//{
    //printf("server received message:%s",a);
    togglecase(a,cnt);
              if(sendto(sockfd,a,50,0,(struct sockaddr*)&cliaddr,sizeof(cliaddr)                    )<0)
                 {
```

```
                printf("error in sendto\n");
                exit(0);
              }
             // }
              //}
               close(sockfd);
        }
           void togglecase(char *a,int cnt)
           {
             int i;
               int k=atoi(a);
               if(k>=60)
   strcpy(a,"you have passed in a grade \n");
               else if((k>=50) && (k<60))
                   strcpy(a,"you have passed in b grade \n");
               else if((k>=35) && (k<50))
                   strcpy(a,"you have passed in c grade \n");
               else
                   strcpy(a,"you have passed in d grade \n");
               //printf("%d",k);
             for(i=0;i<cnt;i++)
             {
                if((a[i] >='A') && (a[i] <= 'Z'))
                   a[i] += 0X20;
                else if((a[i] >='a') && (a[i] <='z'))
                   a[i] -= 0X20;
             }
         }
```

Output:

```
$ vi Gradeserver.c
$ cc -o b Gradeserver.c
$ ./b 8000
server recd the message:80
```

Gradeclient:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
  int main(int argc,char *argv[])
  {
        int sockfd,servlen;
        char a[50],a1[50];
        struct sockaddr_in servaddr,cliaddr;
        sockfd=socket(AF_INET,SOCK_DGRAM,0);
        if(argc!=3)
        {
           printf("usage:<client> <port> <ipaddress>\n");
           exit(0);
        }
        if(sockfd < 0)
        {
                printf("can't bind");
                exit(0);
        }
        servaddr.sin_family=AF_INET;
        servaddr.sin_addr.s_addr=inet_addr(argv[2]);
        servaddr.sin_port=htons(atoi(argv[1]));
        cliaddr.sin_family=AF_INET;
        cliaddr.sin_addr.s_addr=inet_addr(argv[2]);
        cliaddr.sin_port=htons(0);
        bind(sockfd,(struct sockaddr*)&cliaddr,sizeof(cliaddr));
        printf("enter a Percentage:");
        fgets(a,50,stdin);
servlen=sizeof(servaddr);
        sendto(sockfd,a,50,0,(struct sockaddr*)&servaddr,servlen);
        recvfrom(sockfd,a1,50,0,(struct sockaddr*)&servaddr,&servlen);
        printf("client received msg:%s",a1);
}
```

Output:

```
$ cc -o a Gradeclient.c
$ ./a 8000 172.0.5.48
enter a Percentage:80
client received msg:YOU HAVE PASSED IN A GRADE
```

PROGRAM No.08

<u>Aim:</u> Implementation of Distance Vector Routing Protocol.

INTRODUCTION: Routing algorithm is a part of network layer software which is responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagram internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits internally, routing decisions are made only when a new established route is being set up. The latter case is sometimes called session routing, because a rout remains in force for an entire user session (e.g., login session at a terminal or a file).

Routing algorithms can be grouped into two major classes: adaptive and non-adaptive. Non-adaptive algorithms do not base their routing decisions on measurement or estimates of current traffic and topology. Instead, the choice of route to use to get from I to J (for all I and J) is compute in advance, offline, and downloaded to the routers when the network ids booted. This procedure is sometime called static routing.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get information (e.g., locally, from adjacent routers, or from all routers), when they change the routes, and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time).

Two algorithms in particular, distance vector routing and link state routing are the most popular. Distance vector routing algorithms operate by having each router maintain a table (i.e., vector) giving the best known distance to each destination and which line to get there. These tables are updated by exchanging information with the neighbor's. The distance vector routing algorithm is sometimes called by other names, including the distributed Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the RIP and in early versions of DEC net and Novell'Ps IPX. AppleTalk and Cisco routers use improved distance vector protocols.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in subnet. This entry contains two parts: the preferred outgoing line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

The router is assumed to know the "distance" to each of its neighbor. If the metric is hops, the distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special ECHO packets that the receiver just time stamps and sends back as fast as possible.

Algorithm:

1. Start

2. By convention, the distance of the node to itself is assigned to zero and when a node is unreachable the distance is accepted as 999.

3. Accept the input distance matrix from the user (*dm[][]*) that represents the distance between each node in the network.

4. Store the distance between nodes in a suitable variable.

5. Calculate the minimum distance between two nodes by iterating.

6. If the distance between two nodes is larger than the calculated alternate available path, replace

   the existing distance with the calculated distance.

7. Print the shortest path calculated.

8. Stop.

Distance Vector Routing Program:

```
#include<stdio.h>
struct node
{
unsigned dist[20];
unsigned from[20];
}
rt[10];
int main()
{
int dmat[20][20];
int n,i
,j,k,count=0;
printf("\nEnter the number of nodes : ");
scanf("%d",&n); printf("\nEnter the cost matrix :\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
scanf("%d",&dmat[i][j]);
dmat[i][i]=0;
rt[i].dist[j]=dmat[i][j];
rt[i].from[j]=j;
}
do
{
count=0;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
for(k=0;k<n;k++)
if(rt[i].dist[j]>dmat[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<n;i++)
{
printf("\n\nState value for router %d is \n",i+1);
for(j=0;j<n;j++)
{
printf("\t\nnode %d via %d Distance%d",j+1,rt[i].from[j]+1,rt[i].dist[j]);
}
}
printf("\n\n");
}
```

<u>Output:</u>

```
$ cc -o a distance.c
$ ./a
Enter the number of nodes : 2
Enter the cost matrix :
1
3
4
5
State value for router 1 is
node 1 via 1 Distance0
node 2 via 2 Distance3
State value for router 2 is
node 1 via 1 Distance4
node 2 via 2 Distance0
```

PROGRAM No: 09

Aim: Write a Program to perform encryption and decryption using the following techniques.

a. Ceaser cipher **b. Substitution cipher**

Introduction:

The Caesar cipher is one of the earliest known and simplest ciphers. It is a type of substitution cipher in which each letter in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on. The method is named after Julius Caesar, who apparently used it to communicate with his generals.

Example
To pass an encrypted message from one person to another, it is first necessary that both parties have the 'key' for the cipher, so that the sender may encrypt it and the receiver may decrypt it. For the caesar cipher, the key is the number of characters to shift the cipher alphabet.
Here is a quick example of the encryption and decryption steps involved with the caesar cipher. The text we will encrypt is 'defend the east wall of the castle', with a shift (key) of 1.

plaintext: defend the east wall of the castle
ciphertext: efgfoe uif fbtu xbmm pg uif dbtumf

It is easy to see how each character in the plaintext is shifted up the alphabet. Decryption is just as easy, by using an offset of -1.

plain: abcdefghijklmnopqrstuvwxyz
cipher: bcdefghijklmnopqrstuvwxyza

Obviously, if a different key is used, the cipher alphabet will be shifted a different amount.
Mathematical Description

First we translate all of our characters to numbers, 'a'=0, 'b'=1, 'c'=2, ... , 'z'=25. We can now represent the caesar cipher encryption function, e(x), where x is the character we are encrypting, as:

$$e(x) = (x + k) \pmod{26}$$

Where k is the key (the shift) applied to each letter. After applying this function the result is a number which must then be translated back into a letter. The decryption function is :

$$e(x) = (x - k) \pmod{26}$$

(a)Ceaser cipher:

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

// Function to encrypt the text
void encrypt(char text[], int shift) {
    for (int i = 0; text[i] != '\0'; ++i) {
        char ch = text[i];
        if (ch >= 'a' && ch <= 'z') {
            ch = (ch - 'a' + shift) % 26 + 'a';
        } else if (ch >= 'A' && ch <= 'Z') {
            ch = (ch - 'A' + shift) % 26 + 'A';
        }
        text[i] = ch;
    }
}

// Function to decrypt the text
void decrypt(char text[], int shift) {
    for (int i = 0; text[i] != '\0'; ++i) {
        char ch = text[i];
        if (ch >= 'a' && ch <= 'z') {
            ch = (ch - 'a' - shift + 26) % 26 + 'a';
        } else if (ch >= 'A' && ch <= 'Z') {
            ch = (ch - 'A' - shift + 26) % 26 + 'A';
        }
        text[i] = ch;
    }
}

int main() {
    char text[100];
    int shift;

    printf("Enter a message to encrypt: ");
    gets(text);
    printf("Enter shift amount: ");
    scanf("%d", &shift);

    encrypt(text, shift);
    printf("Encrypted message: %s\n", text);

    decrypt(text, shift);
    printf("Decrypted message: %s\n", text);

    return 0;
}
```

```
OUTPUT:              Enter a message to encrypt: abc
                     Enter shift amount: 3

                     Encrypted message: def,     Decrypted message: abc
```

PROGRAM No.10

Aim: Write a program to implement DES Algorithm

An Overview of Symmetric Key Cryptography
Symmetric key cryptography (or symmetric encryption) is a type of encryption scheme in which the same key is used both to encrypt and decrypt messages. Such a method of encoding information has been largely used in the past decades to facilitate secret communication between governments and militaries.

Data Encryption Standard:
The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST). DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit.
The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).

How DES Works?
DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only). General Structure of DES is depicted in the following illustration

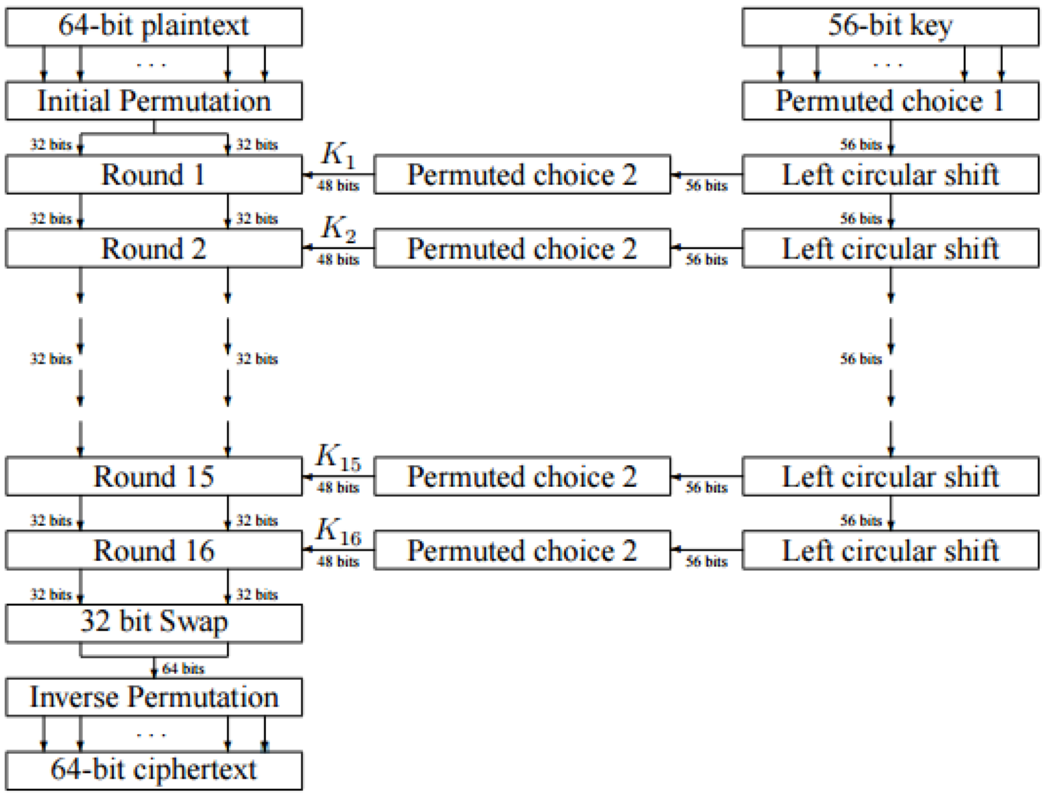


Figure : Flow Diagram of DES algorithm for encrypting data.

So in total the processing of the plaintext proceeds in three phases as can be seen from the left hand side of figure

1. Initial permutation (IP - defined in table 2.1) rearranging the bits to form the “permuted input”.

2. Followed by 16 iterations of the same function (substitution and permutation). The output of the last iteration consists of 64 bits which is a function of the plaintext and key. The left and right halves are swapped to produce the preoutput.

3. Finally, the preoutput is passed through a permutation (IP−1 - defined in table 2.1) which is simply the inverse of the initial permutation (IP). The output of IP−1 is the 64- bit ciphertext

Details of a Single DES Round:

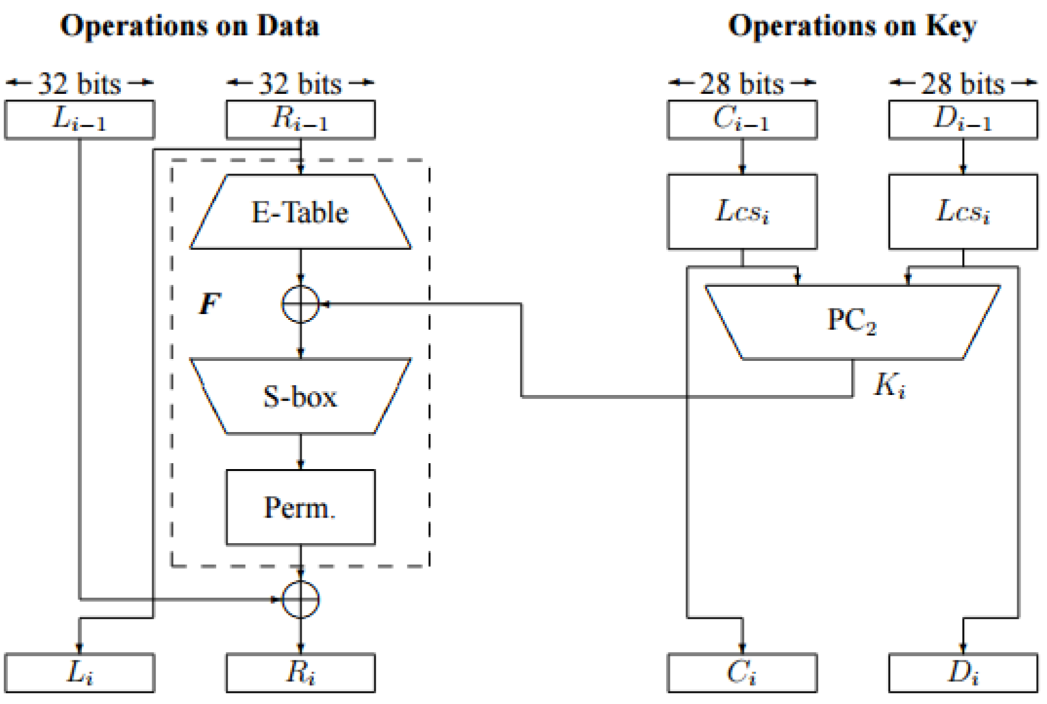


Figure : Details of a single DES round.

DES Program:

```
import javax.swing.*;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Random ;
class DES {
byte[] skey = new byte[1000];
String skeyString;
static byte[] raw;
String inputMessage,encryptedData,decryptedMessage;
public DES() {
try {
generateSymmetricKey();
inputMessage=JOptionPane.showInputDialog(null,"Enter message to encrypt");
byte[] ibyte = inputMessage.getBytes();
byte[] ebyte=encrypt(raw, ibyte);
String encryptedData = new String(ebyte);
System.out.println("Encrypted message "+encryptedData);
JOptionPane.showMessageDialog(null,"Encrypted Data "+"\n"+encryptedData);

byte[] dbyte= decrypt(raw,ebyte);
String decryptedMessage = new String(dbyte);
System.out.println("Decrypted message "+decryptedMessage);

JOptionPane.showMessageDialog(null,"Decrypted Data "+"\n"+decryptedMessage);
}
catch(Exception e) {
System.out.println(e);
}

}
void generateSymmetricKey() {
try {
Random r = new Random();
int num = r.nextInt(10000);
String knum = String.valueOf(num);
byte[] knumb = knum.getBytes();
skey=getRawKey(knumb);
skeyString = new String(skey);
System.out.println("DES Symmetric key = "+skeyString);
}
catch(Exception e) {
System.out.println(e);
}
}
private static byte[] getRawKey(byte[] seed) throws Exception {
KeyGenerator kgen = KeyGenerator.getInstance("DES");
SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
sr.setSeed(seed);
kgen.init(56, sr);
SecretKey skey = kgen.generateKey();
raw = skey.getEncoded();
return raw;
}
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES");
Cipher cipher = Cipher.getInstance("DES");
cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
byte[] encrypted = cipher.doFinal(clear);
return encrypted;
}

private static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception {
SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES");
Cipher cipher = Cipher.getInstance("DES");
cipher.init(Cipher.DECRYPT_MODE, skeySpec);
byte[] decrypted = cipher.doFinal(encrypted);
return decrypted;
}
public static void main(String args[]) {
DES des = new DES();
}
}
```

OUTPUT:

```
Enter the string: Welcome
String To Encrypt: Welcome
Encrypted Value : BPQMwc0wKvg=
Decrypted Value : Welcome
```

PROGRAM No: 11
<u>Aim:</u> Implementation of RSA public key algorithm.

INTRODUCTION:
Cryptography has a long and colourful history. The message to be encrypted, known as the plaintext, are transformed by a function that is parameterized by a key. The output of the encryption process, known as the cipher text, is then transmitted, often by messenger or radio. The enemy, or intruder, hears and accurately copies down the complete cipher text. However, unlike the intended recipient, he does not know the decryption key and so cannot decrypt the cipher text easily. The art of breaking ciphers is called cryptanalysis the art of devising ciphers (cryptography) and breaking them (cryptanalysis) is collectively known as cryptology. There are several ways of classifying cryptographic algorithms. They are generally categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms are as follows:

1. Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption. It is also known as symmetric cryptography.

2. Public Key Cryptography (PKC): Uses one key for encryption and another for decryption. It is also known as asymmetric cryptography.

3. Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information.

Public-key cryptography has been said to be the most significant new development in cryptography. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key.

Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the cipher text. The important point here is that it does not matter which key is applied first, but that both keys are required for the process to work. Because pair of keys is required, this approach is also called asymmetric cryptography.

In PKC, one of the keys is designated the public key and may be advertised as widely as the owner wants. The other key is designated the private key and is never revealed to another party. It is straight forward to send messages under this scheme.

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

<u>Algorithm:</u>

1. Generate two large random primes, P and Q, of approximately equal size.

2. Compute N = P x Q

3. Compute Z = (P-1) x (Q-1).

4. Choose an integer E, 1 < E < Z, such that GCD (E, Z) = 1

5. Compute the secret exponent D, 1 < D < Z, such that E x D 1 (mod Z)

6. The public key is (N, E) and the private key is (N, D).

Note: The values of P, Q, and Z should also be kept secret. The message is encrypted using public key and decrypted using private key.

An example of RSA encryption

1. Select primes P=11, Q=3.

2. N = P x Q = 11 x 3 = 33 Z = (P-1) x (Q-1) = 10 x 2 = 20

3. Lets choose E=3 Check GCD(E, P-1) = GCD(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1), and check GCD(E, Q-1) = GCD(3, 2) = 1 therefore GCD(E, Z) = GCD(3, 20) = 1

4. Compute D such that E x D 1 (mod Z)

compute D = E ^-1 mod Z = 3 ^-1 mod 20

find a value for D such that Z divides ((E x D)-1)

find D such that 20 divides 3D-1.

Simple testing (D = 1, 2, ...) gives D = 7

Check: (E x D)-1 = 3.7 - 1 = 20, which is divisible by Z.

5. Public key = (N, E) = (33, 3)Private key = (N, D) = (33, 7).

Now say we want to encrypt the message m = 7,
Cipher code = M ^E mod N = 7 ^3 mod 33
= 343 mod 33 = 13. Hence the cipher text c = 13.
6. To check decryption we compute Message' = C ^D mod N = 13 ^7 mod 33
7. Note that we don't have to calculate the full value of 13 to the power 7 here. We can make use of the fact that a = bc mod n = (b mod n).(c mod n) mod n so we can break down a potentially large number into its components and combine the results of easier, smaller calculations to calculate the final value.

RSA Program:

```
#include<stdio.h>
int phi,M,n,e,d,C,FLAG;
int check(int e, int phi)
{
int i;
for(i=3;(e%i==0 && phi%i==0);i+2)
{
FLAG = 1;
return;
}
FLAG = 0;
}
void encrypt()
{
int i;
C = 1;
for(i=0;i< e;i++)
{
C=C*M%n;
}
C = C%n;
printf("\n\tEncrypted keyword : %d",C);
}
void decrypt()
{
int i;
M = 1;
for(i=0;i< d;i++)
{
M=M*C%n;
}
M = M%n;
printf("\n\tDecrypted keyword : %d\n",M);
}
void main()
{
int p,q,s;
printf("Enter Two Relatively Prime Numbers\t: ");
scanf("%d%d",&p,&q); /*let 7 & 17*/
n = p*q;
/* this is the modules for both public and private key */
phi=(p-1)*(q-1);
/*f( n ) = (p–1)(q–1), where f is Euler's totient function*/
printf("\n\tF(n)\t= %d",phi);
do
{
printf("\n\nEnter e\t: ");
/* enter an integer e(5) such that 1 < e < f(n)
and gcd(e,f(n)) = 1, i.e. e and f(n) are coprime*/
scanf("%d",&e);
check(e, phi);
}
while(FLAG==1);
/*at the end of this do while, we will get the value of e*/
d = 1;
/*Determine d = e^–1 mod f(n); */
do
{
s = (d*e)%phi;
d++;
} while(s!=1);
d = d-1; /*we got the value of d*/
printf("\n\tPublic Key\t: {%d,%d}",e,n);
/* public key consists of the modulus n(p*q)
and the public (or encryption) exponent e*/
printf("\n\tPrivate Key\t: {%d,%d}",d,n); /* private key
decryption)
exponent d which must be kept secret*/
printf("\n\nEnter The Plain Text\t: ");
scanf("%d",&M);
encrypt();
printf("\n\nEnter the Cipher text\t: ");
scanf("%d",&C);
decrypt();
}
```

Output:

```
$ cc -o aa rsa.c

$ ./aa



Enter Two Relatively Prime Numbers     : 7 5

        F(n)     = 24

Enter e  : 7

        Public Key        : {7,35}

        Private Key       : {7,35}

Enter The Plain Text      : 45

        Encrypted keyword : 10

Enter the Cipher text     : 10     Decrypted keyword : 45
```

PROGRAM No: 13

AIM: Calculate the message digest of a text using the SHA-1 algorithm

```
//---------------SecureHashAlgorithm-1(SHA-1)----------------//

 import java.security.*;
 public class SHA1 {
 public static void main(String[] a) {
 try {
 MessageDigest md = MessageDigest.getInstance("SHA1");
System.out.println("Message digest object info: ");
System.out.println(" Algorithm = " +md.getAlgorithm());
System.out.println(" Provider = " +md.getProvider());
System.out.println(" ToString = " +md.toString());
String input = "";
 md.update(input.getBytes());
 byte[] output = md.digest();
 System.out.println();
 System.out.println("SHA1(\""+input+"\") = " +bytesToHex(output));
 input = "abc";
 md.update(input.getBytes());
 output = md.digest();
 System.out.println();
 System.out.println("SHA1(\""+input+"\") = " +bytesToHex(output));
 input = "abcdefghijklmnopqrstuvwxyz";
 md.update(input.getBytes());
 output = md.digest();
 System.out.println();
System.out.println("SHA1(\"" +input+"\") = " +bytesToHex(output));
System.out.println(""); }
catch (Exception e) {
System.out.println("Exception: " +e);
}
}
public static String bytesToHex(byte[] b) {
char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
StringBuffer buf = new StringBuffer();
for (int j=0; j<b.length; j++) {
buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
buf.append(hexDigit[b[j] & 0x0f]); }
return buf.toString(); }
}
```

Output:

PROGRAM No: 14

AIM: Calculate the message digest of a text using the MD5 algorithm.

```
import java.math.BigInteger;

  import java.security.MessageDigest;

  import java.security.NoSuchAlgorithmException;

  public class JavaMD5Hash {

        public static void main(String[] args) {

                System.out.println("For null " + md5(""));

                System.out.println("For simple text "+ md5("This is my text"));

                System.out.println("For simple numbers " + md5("12345"));

        }

        public static String md5(String input) {

                String md5 = null;

                if(null == input) return null;

                try {

                        //Create MessageDigest object for MD5

                        MessageDigest digest = MessageDigest.getInstance("MD5");

                        //Update input string in message digest

                        digest.update(input.getBytes(), 0, input.length());

                        //Converts message digest value in base 16 (hex)

                        md5 = new BigInteger(1, digest.digest()).toString(16);

                }

                catch (NoSuchAlgorithmException e) {

                        e.printStackTrace();

                }

                return md5;

        }   }
```

Output:

PROGRAM No. 15:

AIM: To create simple topology using Network Simulator – 2.

ALGORITHM:

Step 1: Start network simulator OTCL editor.

Step 2: Create new simulator using set ns [new Simulator] syntax Step 3: Create Trace route to Network Animator

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

Step 4: Create procedure to trace all path

```
proc create_testnet {} { global s1 s2 r1
k1 set s1 [$ns node]
 set  s2  [$ns
 node]  set  r1
 [$ns    node]
 set  k1  [$ns
 node]
}
```

Step 5: Create full duplex connection using

```
$ns duplex-link $s1 $r1 8Mb 5ms drop-tail
$ns duplex-link $s2 $r1 8Mb 5ms drop-tail
set L [ns_duplex $r1 $k1 800Kb 100ms drop-
tail] Step 4: Connect with TCP and SINK
command.
$ns connect $tcp $sink
```

Step 5: Run and Execute the program.

```
$ns run
```

PROGRAM:

```
set ns [new
Simulator] set nf
[open udp.nam w]
$ns namtrace-all
$nf set tf [open out.tr w]
$ns trace-all $tf
proc create_testnet {} { global s1 s2 r1 k1
                    set  s1  [$ns
                   node]  set  s2
                   [$ns    node]
                   set  r1  [$ns
                   node]  set  k1
                   [$ns node]
}
$ns duplex-link $s1 $r1 8Mb 5ms drop-tail
$ns duplex-link $s2 $r1 8Mb 5ms drop-tail
set L [ns_duplex $r1 $k1 800Kb 100ms drop-tail] [lindex $L 0] set queue-limit 6
[lindex $L 1] set queue-limit 6
$ns run
```

OUTPUT:

RESULT:

CN VIVA Questions

What is the result for command “ifconfig” ?
Ifconfig –arp what is the use of this command?
Ping 172.168.1.22 what is output format ?

1. What is the use of netstat?
2. Netstat -r output ?
3. Telnet how it useful in communication ?
4. What is socket ?
5. What is active socket?
6. What is passive socket ?
7. Which socket use Bind() ?
8. A passive socket use Bind() and active cannot why?
9. Which socket use connect() active/passive ?
10. Which socket use listen() client/server(or) active/passive ?
11. What is connection queue ?
12. What are the socket constants?
13. AF_INET when we will use ?
14. AF_UNIX when we will use ?
15. A communication between lab system to home system I want to establish then which address family I want to use ?
16. In a mini project in my system 2 nodes want to communicate which family I have to use.
17. When we will use PF_INET6 ?
18. Arguments for Socket()?
19. What is the use of SOCK_STREAM ?
20. What is the use of SOCK_DGRAM ?
21. Output of the program

```
simpleSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP); if
(simpleSocket == -1) {
fprintf(stderr, "Could not create a socket!\n");
exit(1);}
else { fprintf(stderr, "Socket created!\n"); }
```

22. Output of the program

```
simpleSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_UDP); if
(simpleSocket == -1) {
fprintf(stderr, "Could not create a socket!\n");
exit(1);}
```

```
else { fprintf(stderr, "Socket created!\n"); }
```

23. Output of the program

```
simpleSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_TCP); if
(simpleSocket == -1) {
fprintf(stderr, "Could not create a socket!\n"); exit(1);}
else { fprintf(stderr, "Socket created!\n"); }
```

24. Output of the program

```
simpleSocket = socket(PF_INET6, SOCK_STREAM, IPPROTO_TCP); if
(simpleSocket == -1) {
fprintf(stderr, "Could not create a socket!\n"); exit(1);}
else { fprintf(stderr, "Socket created!\n"); }
```

25. Which Protocol accept by default Socket = socket(AF_INET, SOCK_STREAM, 0);
26. Which Protocol accept by default Socket = socket(AF_INET, SOCK_DGRAM, 0);
27. What the socket written on creation as output ?
28. What is socket file descriptor ?
29. Arguments for Bind() ?
30. What is the use of Bind() ?
31. simplePort = atoi(argv[1]); Means ?
32. What are the structure variables inAddress Structure?
33. What is the use of use INADDR_ANY ?
34. What is the use of hton() ?
35. What is the use of ntoh()?
36. What is network byte ordering ?
37. No of octets accepted by htonl() ?
38. No of octets accepted by htons() ?
39. If socket value is -1 what is the output

```
returnStatus = bind(simpleSocket, (structsockaddr *)&simpleServer,  sizeof(simpleServer)); if
        (returnStatus == 0)
       { fprintf(stderr, "Bind completed!\n"); }
       else { fprintf(stderr, "Could not bind to address!\n");
       close(simpleSocket);
        exit(1); }
```

40. If socket value is an integer what is the output

```
returnStatus = bind(simpleSocket, (structsockaddr *)&simpleServer,  sizeof(simpleServer)); if
        (returnStatus == 0)
        { fprintf(stderr, "Bind completed!\n"); }
        else { fprintf(stderr, "Could not bind to address!\n");
        close(simpleSocket);
         exit(1); }
```

41. What will be the output of Bind() on successful bind of address ?
42. What is the use of listen () ?
43. Arguments accepted by listen() ?
44. What is the need of backlog value in Listen() ?
45. The maximum value of backlog in BSD based systems ?

46. If s=-1& backlog =5 what will be the output

```
  returnStatus = listen(s, backlog);
    if (returnStatus == -1)
      { fprintf(stderr, "Cannot listen on socket!\n");} else
      { fprintf(stderr, "Ready to listen!\n");
close(simpleSocket);
  exit(1); }
```

47. If Socket was successfully created& backlog =5 what will be the output

```
  returnStatus = listen(s, backlog);
    if (returnStatus == -1)
      { fprintf(stderr, "Cannot listen on socket!\n"); } else {
      fprintf(stderr, "Ready to listen!\n");
close(simpleSocket);
  exit(1); }
```

48. What is the use of accept () ?
49. Arguments accepted by accept ()?
50. How many times the accept executes in server side?
51. Which is called as blocking function and why?
52. What is the o/p generated on successful accept()?
53. How connect() works ?
54. The arguments of connect function?
55. What is difference b/w bind() and accept() ?

56. The o/p connect function on successful and failure?
57. Where we use tcp and udp ?
58. What is the use MAX_BUFF constant?
59. What is the use of recvfrom() ?
60. If the buffer size is 1024 but am sending the packet size is 2048 to server what will happen /
61. What is the recvfrom() output on successful &failed ?
62. What are the 2 blocking functions will put our process into sleep while in communication ?
63. What is the major difference b/w concurrent iterative server in process level ?
64. Which command majorly used in process level for clients purpose to handle their requests inn concurrent server ?
65. How seclect () system call will useful in multithreaded server?
66. FD_ZERO what is use of this function?
67. FD_SET what is use of this function ?
68. FD_CLR() what is use of this function ?
69. FD_ISSET() what is use of this function ?
70. What is *select()* function prototype ?
71. What is the use of getsockopt() ?
72. What is the use of setsockopt() ?
73. Syntax for setsockopt() ?
74. Syntax for getsockopt ?
75. What is the use of SO_SNDBUF?
76. What is the output of getsocket() and set socket on successful execution?
77. What is the difference between getsockname() and getpeername()?
78. getsockname() and getpeername() how they are different in calling sockaddr structure?
79. what is the necessity ofgetpeername() function in server side ?
80. syntax of getpeername() ?
81. syntax of getsockname() ?
82. what is the use of system() systemcall ?
83. what is the use of fflush() function ?
84. which directory by default accessed by System() ?
85. Which algorithm used by distance vector routing algorithm by default?
86. Distance vector routing algorithm what it will do in each node?
87. How we find the optimal path?
88. What is smtp?

Viva Questions :

1.what is another name for Rail fence cipher?

2.howto represent the palin text in Rail fence cipher?

3.what is ciphertext ofpalin text "INCLUDEHELP IS AWESOME"?

4.How many messages used in SSL handshake protocol?

5.write an example of passive attack?