ADVANCE COMPUTER NETWORKS

LAB MANUAL

For

IV SEMESTER

INDIRA GANDHI DELHI TECHNICAL UNIVERSITY FOR WOMEN

Kashmere Gate, Delhi

Goals:

Get familiar with the C/C++ and JAVA Programming Language: If you don't already know it, in this lab you will get started on learning a new programming language. Although this is not meant to be a tutorial, you will get exposed to the most basic features of the language by reading and analyzing some basic programs.

Learn to use or get better practice with Unix *system calls*: Operating Systems provide programmers with a range of services that can be used directly in their applications. These services are most often presented at *system calls*, that is, functions that applications can use to ask the operating systems to perform certain tasks for them. Since this course is based on the Unix operating system, we will be working exclusively with Unix system calls. These services are commonly implemented in roughly the same way across different flavors of Unix. At Bucknell, we currently work with a mixture of Sun Solaris and Intel Linux machines, so beware: depending on the platform you work there may be small differences in how these system calls are implemented. The Unix manual pages available on the computing platform will be of great help to you in understanding what specific system calls can do for your application and in how you use them. Make sure to take this opportunity to learn to navigate the manual pages effectively if you don't already know how to do that.

The C Programming Language

In this lab you will spend some time getting acquainted with the C programming language. Rather than provide you with a step-by-step tutorial in the language, we will discuss the most significant differences from C++, which you're assumed to know. For the experienced C++ programmer, the C language will look much more low-level, as it indeed is. We discuss some of the features of the language in more detail later on in this lab, but point out the most salient differences you will encounter in C coming from C++:
There are no classes. C is not an object-oriented language.
Comments must be enclosed in ***/* ... */***, single-line comment ***//*** is supported only in the more recent versions of C.
The nesting of comment blocks is not allowed.
Strings are simply null-terminated arrays of **char**.
Type checking for function arguments may be quite lax, meaning that more burden is on the programmer.
There isn't stream-based input/output (cin, cout). The available operations for I/O seem crude because they are lower-level.
Dynamic memory allocation is done via library functions such as **malloc**, **calloc**, and **free** (no **new** / **delete**). You allocate and deallocate arrays of bytes instead of "*space for a data-type*".
Parameter passing is exclusively by *value*. If you need to modify a parameter passed into a function, you must pass the pointer to the parameter.

There exist "dialects" of the C language, which define some minor syntactical differences and alternatives to type-checking. The dialect used depends, of course, on the compiler you use. Three of the most common dialects are:
Kernighan & Ritchie C (or simply K&R C, named after the creators of the language) .
ANSI C (which has is much stricter about type-checking than K&R C).

The BSD Socket API

Sockets are like Unix pipes in that they use file descriptors to define the endpoints of a communication channel. When you make the system call to create a pipe, you pass in the pointer to an integer array with two elements. The successful competion of the pipe call returns 0 and initializes the array with two file descriptors, one to which you can write and one from which you can read. The socket follows a more complex set up process which we describe below.

Creating a socket

```
#include <sys/types.h>

 #include <sys/socket.h>

 int socket(int domain, int type, int protocol);
```

The socket call creates one single endpoint for communications, unlike pipe which returns two. If the value returned is less than 0, the call was unsuccessful and the error can be discovered through **errno** and **perror**. If the value returned is greater than or equal to 0, the call was successful and the returned value is a File-descriptor.

Althouth the AF_UNIX domain adds some interesting and flexible options for IPC within the same host, we will leave it lone for now. We are more interested in domains that allow two hosts connected to the Internet to communicate and will focus exclusively on the AF_INET for now.

The next parameter, type, can assume one of three possible values as described below:

SOCK_DGRAM: provides *datagram* communication semantics, that is, this is a connectionless protocol that will do its best to deliver the data without making any promises that the service is reliable. There are no guarantees that the datagrams are delivered in the order they are sent.

SOCK_STREAM: provices a bidirectional *virtual circuit* that is reliable and order-preserving (FIFO).

SOCK_RAW: provides the ability to send packets directly to a network device driver, enabling user space applications to provide networking protocols that are not implemented in the kernel.

The last parameter, protocol, can be left unspecified (value 0) so that the default protocol that supports the protocol family and socket type is used. In case multiple protocols exist for the selected tuple <domain, type>, then a specific protocol must be specified. After the successful completion of the socket call, what you have is a connection endpoint that is not attached anywhere. Before any communication can happen, the socket must be associated to "something". The process of connecting the socket is an asymmetric task, that is, it is performed differently at each endpoint of the socket. Server applications (which run on "infinite loops"), create the socket, get it ready to be hooked up to something, and then wait for someone to request a connection to the socket. Client processes, on the other hand, create a socket, tell the system to which address they want to connect it, and attempt establishing the connection to the server. The server

process then accepts the connection request and the socket is finally ready for communication.

Binding an address to a socket

Before we proceed, you need to understand what an Internet address and how it is represented in Unix. Header file <netinet/in.h> defines a 32-bit address for an Internet host. It actually identifies a specific network interface on a specific system on the Internet using the data structure below:

```
struct in_addr {
__u32 s_addr;
};
```

Demonstration of How a Header File works:

```
/************************************************************** * * *
   decls.h is a demonstration file to show how header file works.* * *
   ***************************************************************/
int i; /* a integer variable */
float x; /* a float variable */
```

****

```
/************************************************************** * * *
                              foo.c

   * A simple C module file that contains a function called 'foo' * * *
    ***************************************************************/
#include "decls.h"
 foo(int k) {
 printf("i = %d, k = %d\n", i, k);
}
****************** /*****************************************************
main()

 The main() function that calls the function 'foo()' * * *
*****************************************************/
 #include "decls.h"
int main(void)
 {
 i = 1;
 if (i == 0)
x = 1000;
else foo(-1);
printf(" result x = %f\n", x);
return 0;
}
```

1. Syllabus

As taken from www.ipu.ac.in website*.

**Academics → Academics → Scheme Syllabus (Affiliated Institutes, w.e.f. 2005-06)*

UNIT - I

Introduction: Overview of computer network, seven- layer architecture, TCP/IP suite of protocol, etc, Mac protocol for high speed LANS, MAN's & WIRLESS LANs (for example, FDDI, DQDB, HIPPI, Gigabit Ethernet, Wireless Ethernet etc) Fast access technologies.(For example, ADSL, cable Modem Etc.), Wi Fi, Wimax. [**No. of hrs: 10]**

UNIT – II

IPV6: Why IPV6, basic protocol, extension & option, support for QS, Security, etc, neighbor discover, auto-configuration, routing, Change to other protocols, Application programming interface for IPV6.6 bone. **ATM:** Introduction, ATM reference Model, AAL layers, AAL0, AA1, AAL2, AAL3/4, AAL5. [**No. of hrs: 12]**

UNIT – III

Mobility in network, mobile, Security related issues. **IP Multicasting**: Multicasting routing protocols, address assignment, session discovery, etc. [**No. of hrs: 10]**

UNIT-IV

TCP extensions for high – speed networks, transaction – oriented application, other new option in TCP. **Network security at various layers**: Secure-HTTP, SSP, ESP, Authentication header, key distribution protocols, Digital signatures, digital certificates.

[**No. of hrs: 10]**

TEXT BOOKS:

1. W. ER. Stevens, "TCP/IP illustrated, Volume 1: The protocols", Addison Wesley, 1994.
2. G. R. Wright, "TCP/IP illustrated volume 2. The Implementation", Addison Wesley, 1995.
3. Frouzan, "TCP/IP Protocol Suite", Tata Mc Grew Hill, 4th Ed., 2009.

REFERENCES:

1. William Stalling, "Cryptography and Network Security", Pearson Publication.
2. James Martin, Joseph Lebin, Kavanagh Chapman "Asynchronous Transfer Mode: ATM Architecture and Implementation", Prentice Hall PTR, Facsimile Ed.

Lesson Plan
Advanced Computer Networks: MCA-206

Number of Theory Hours per week: 4 hrs		
Books Recommended		
Primary Text Book	Behrouz A. Forouzan, "TCP/IP Protocol Suit", TMH, 4th Edition Behrouz A. Forouzan, "Data Communication and Networking", TMH, 4rd Edition	[ACN] [DCN]
Reference Book	Data Communications and Computer Networks, Prakash C. Gupta, PHI Computer Networks and Internets, Douglas E. Comer, M.S. Narayanan, Pearson	[PCG] [DEC]

Topic wise Schedule

Topic	Book	Duration
Introduction : Overview of computer network, seven- layer architecture, TCP/IP suite of protocol, etc,	[ACN]	2 Hrs
Mac protocol for high speed LANS, MAN's & WIRLESS LANs (for example, FDDI, DQDB, HIPPI, Gigabit Ethernet, Wireless Ethernet etc)	[ACN]	2 Hrs
Fast access technologies.(For example, ADSL, cable Modem Etc.)	[DEC]	2 Hrs
Wi Fi	[DEC]	1 Hrs
Wimax	[DEC]	1 Hr
IPV6: Why IPV6, basic protocol, extension & option, support for QS, Security, etc, neighbor discover, auto-configuration, routing, Change to other protocols, Application programming interface for IPV6.6 bone.	[ACN]	6 Hrs

Topic	Book	Duration
ATM: Introduction, ATM reference Model, AAL layers, AAL0, AA1, AAL2, AAL3/4, AAL5	[DCN]	4 Hrs
Mobility in network, mobile	[PCG]	2 Hrs
Security related issues		2 Hrs
IP Multicasting: Multicasting routing protocols, address assignment, session discovery, etc.	[ACN]	5 Hrs
TCP extensions for high – speed networks, transaction – oriented application, other new option in TCP.	[ACN]	4 Hrs
Network security at various layers: Secure-HTTP, SSP, ESP, Authentication header, key distribution protocols,	[ACN]	3 Hrs
Digital signatures, digital certificates.	[ACN]	2 Hrs
Total		**36 Hrs**

2. Lab Work Guidelines and Plan

Guiding Principle:

The only way to learn a programming language is to write programs in it solving various problems. The more we practice and get our hands dirty, the more we learn. And as we move on to solve increasingly challenging problems, we start *enjoying* programming.

2.1 Problem Evaluation Plan and Submission Guidelines

The underlying guideline for programming problem submission and evaluation is to minimize the overhead for the students (so that they concentrate on solving problems) and to maximize the continuous assessment.

Following are the modalities in view of the above guiding principle:-

1. All programs for solving a programming problem will be submitted **online to a web server** hosted in the Lab.

2. Programs submitted will undergo *online* compilation and *offline* (optional) testing along with necessary *plagiarism checks* in order to evaluate the program from time to time.

3. Students would maintain a handwritten *lab file* to keep a record of their *weekly progress*. For each programming problem, writing the problem statement and **only the most relevant code snippet** is required to be written in the *lab file*.

4. There would be **zero tolerance** for plagiarism of any kind. In any event of detection of plagiarized code, all students involved would be penalized including the perceived "original" writer. It is responsibility of a student to secure their code.

5. Grades would fall under following categories:

    a. Excellent: EX, students who have been regularly completing all the programming problems (including all advanced and bonus problems), and not just restricting themselves to minimum criteria as mentioned in Section 3.1.

    b. Very Good: A, students who have been regularly completing most of the programming problems (including all advanced and bonus problems), and not just restricting themselves to minimum criteria as mentioned in Section 3.1.

    c. Good: B, students who have been regularly completing all of the programming problems as per the minimum criteria and occasionally solving bonus problems.

    d. Average: C, students who have been almost regularly completing most of the programming problems as per the minimum criteria.

    e. Poor: D, students who have been irregularly completing most of the programming problems as per the minimum criteria as mentioned in Section 3.1.

6. Range of marks (in percentage) corresponding to above mentioned grades are *tentatively* as follows:

Grade	Range of Marks	Remarks
EX	95-100	Excellent
A	85-94	Very Good
B	75-84	Good
C	65-74	Average
D	55-64	Poor

3. Lab Exercises

This section includes a consolidated list of programming exercises.

1. Problem:

Execute the following Network Oriented Commands (with all their options) and observe their Output :

1) **PING**
2) **TRACERT**
3) **ROUTE**
4) **IPCONFIG**
5) **ARP**
6) **NETSTAT**
7) **NBTSTAT**
8) **HOSTNAME**
9) **NETSEND**
10) **DNS Configuration**
11) **Try such 25 more networking commands**

2. Problem:

To write a program for IP addressing classification problem.

PROGRAM DESCRIPTION: In this program based on the type of classification we divide the IP addresses. The user gives the input address based on conclusions st displays the category of the address.

LOGIC:

1.The addresses are classified as per the rules.
2. The input from the user is taken in the form of address.
3. The program compares the first 8 bits of the address and classifies it according to the respective class.

INPUT: Enter the address:127.254.254.254
OUTPUT: The address is a class A Address.

INPUT: Enter the address:192.234.254.224
OUTPUT: The address is a class C Address.

3. Problem:

Write a program of bit stuffing used by Data Link Layer.

PROGRAM DESCRIPTION

Each frame begins and ends with a special bit pattern, 01111110 (in fact, a flag byte). Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This bit stuffing is analogous to byte stuffing, in which an escape byte is stuffed into the outgoing character stream before a flag byte in the data.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit. Just as byte stuffing is completely transparent to the network layer in both computers, so is bit stuffing. If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver's memory as 01111110. Below Figure gives an example of bit stuffing.
With bit stuffing, the boundary between two frames can be unambiguously recognized by the flag pattern. Thus, if the receiver loses track of where it is, all it has to do is scan the input for flag sequences, since they can only occur at frame boundaries and never within the data.

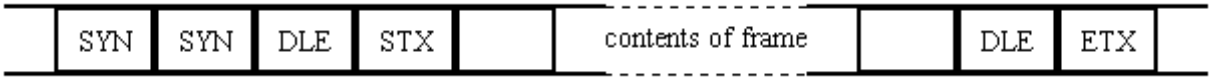
Figure : Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

PSEUDO CODE: /* For given data */
1. a flag “01111110” is embedded at the starting and the ending of the data. /* stuffing of data */
2. if data bit is 1 increment count else count is zero.
3. If count is five store a zero bit after the five 1‟ s in the data array.
4. Repeat step 3 till the end of data. /* De stuffing of data */
5. If the data bit is 1 increment count else count is zero.
6. If the count is five and the next bit is zero then store the next bit after zero in the data array. /* transmit the data */
7. De stuffed data is transmitted with out flags.

At sender:
INPUT: Enter the string 1111110101
OUTPUT: Transmitted data is: 011111101111110101011111110
Stuffed data is: 011111101111101010101111110
At Receiver:
OUTPUT: De stuffed data is: 11111110101
OR
OUTPUT:
Enter frame length: 10
Enter input frame (0's & 1's only):
1 0 1 0 1 1 1 1 1 1
After stuffing the frame is: 1 0 1 0 1 1 1 1 1 0 1

4. Problem:
Write a program of character stuffing used by Data Link Layer.

PROGRAM DESCRIPTION:
A typical frame of a character orientated synchronous protocol is:



The sequence is:

1. The frame is preceded by a number of special synchronisation characters (SYN - value 0010110). The receiver 'hunts for this bit pattern and, when detected, becomes character synchronised with the transmitter and reads contiguous sequences of 7 or 8 bit characters.
2. The start of a frame is signalled by the character sequence DLE (Data Link Escape - value 00010000) STX (Start of Text - value 00000010).
3. The contents of the frame follows as determined by the data link protocol concerned.
4. The end of a frame is signalled by the character sequence DLE ETX - (End of Text - value 00000011).

The problem is that the frame may contain the character sequence *DLE ETX* and prematurely terminate the frame. To overcome this problem a technique called character (or byte) stuffing is used:

Transmitter Examines the character stream to be transmitted for a DLE character and, if found, inserts a second DLE into the output stream.

Receiver Examines the incoming character stream; if a DLE is found it examines the next character, if it is:
DLE it deletes it
ETX is the end of the frame
Anything else is an error.

INPUT:
enter string:
asdlefgh
enter position: 8
invalid position,enter again: 3
enter the character: k

OUTPUT:
String after stuffing:
dlestx as dle k dle dle dlefgh dleetx

5. Problem:
Write a program to implement LRC method (Single-Parity Check)

PROGRAM DESCRIPTION:

A longitudinal redundancy check (LRC) is an error-detection method for determining the correctness of transmitted and stored data.

LRC verifies the accuracy of stored and transmitted data using parity bits. It is a redundancy check applied to a parallel group of bit streams. The data to be transmitted is divided into transmission blocks into which additional check data is inserted.

This term is also known as a horizontal redundancy check.

LRC generally applies to a single parity bit per bit stream. Although simple longitudinal parities only detects errors, a combination with additional error control coding, such as a transverse redundancy check, are capable of correcting errors.

LRC fields consist of one byte containing an eight bit binary value. LRC values are calculated by transmitting devices, which append LRC to messages. The device at the receiving end recalculates the LRC on receipt of the message and compares the calculated value to the actual value received in the LRC field. If the values are equal, the transmission was successful; if the values are not equal, this indicates an error.

LRC is generated through the following steps:

1. Add all bytes in messages excluding the starting colon and the ending the carriage return line feed
2. Add this to the eight-bit field and discard the carries
3. Subtract the final field value from FF hex, producing one's complement
4. Add one, producing two's complement

In a system environment where a data stream is accepted from a host during host-initiated operations, LRC calculations are performed and appended to every received data block. The resulting blocks are stored by the subsystems. As data passes through the subsystem, LRC calculations are performed. If the host requests data later, a data block is sought along with the previously calculated LRC. The same LRC exclusive or calculations are performed and compared with stored LRC values as data is transferred to the host. If the stored value matches the newly calculated values, the data is considered to be valid.

OUTPUT

```
Enter the length of data stream: 10
Enter the data stream 1 1 0 1 0 1 1 1 0 1
Number of 1's are 7
Enter the choice to implement parity bit
1-Sender side
2-Receiver side
1

The data stream after adding parity bit is
11010111011

Enter the length of data stream: 10
Enter the data stream 1 1 1 1 1 0 0 0 1 0
Number of 1's are 6
Enter the choice to implement parity bit
1-Sender side
2-Receiver side
2
There is no error in the received data stream
```

6. Problem:

Write a program for error detecting code using CRC.

PROGRAM DESCRIPTION:
It does error checking via polynomial division. In general, a bit string

$b_{n-1}b_{n-2}b_{n-3}\ldots b_2b_1b_0$

As

$b_{n-1}X^{n-1} + b_{n-2}X^{n-2} + b_{n-3}X^{n-3} + \ldots b_2X^2 + b_1X^1 + b_0$
Ex: -
10010101110

As

$X^{10} + X^7 + X^5 + X^3 + X^2 + X^1$

All computations are done in modulo 2

Algorithm:-

1. Given a bit string, append 0^S to the end of it (the number of 0^s is the same as the degree of the generator polynomial) let B(x) be the polynomial corresponding to B.
2. Divide B(x) by some agreed on polynomial G(x) (generator polynomial) and determine the remainder R(x). This division is to be done using Modulo 2 Division.
3. Define T(x) = B(x) –R(x)

   (T(x)/G(x) => remainder 0)

4. Transmit T, the bit string corresponding to T(x).
5. Let T' represent the bit stream the receiver gets and T'(x) the associated polynomial. The receiver divides $T^1(x)$ by G(x). If there is a 0 remainder, the receiver concludes T = T' and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission.

Output

Enter poly : 1011101
Generating Polynomial is : 10001000000100001
Modified t[u] is : 10111010000000000000000
Checksum is : 1000101101011000
Final Codeword is : 10111011000101101011000
Test Error detection 0(yes) 1(no) ? : 0
Enter position where you want to insert error : 3
Errorneous data : 101**0**10110001011010110000
Error detected.

Enter poly : 1011101
Generating Polynomial is : 10001000000100001
Modified t[u] is : 10111010000000000000000
Checksum is : 1000101101011000
Final Codeword is : 10111011000101101011000

Test Error detection 0(yes) 1(no) ? : 1
No Error Detected.

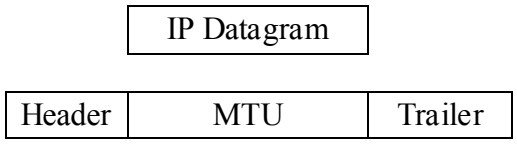
7. Problem:
Write a program for frame sorting technique used in buffers.

PROGRAM DESCRIPTION:

The data link layer divides the stream of bits received from the network layer into manageable data units called frames.

If frames are to be distributed to different systems on the network, the Data link layer adds a header to the frame to define the sender and/or receiver of the frame.

Each Data link layer has its own frame format. One of the fields defined in the format is the maximum size of the data field. In other words, when datagram is encapsulated in a frame, the total size of the datagram must be less than this maximum size, which is defined by restriction imposed by the hardware and software used in the network.



The value of MTU differs from one physical network to another

In order to make IP protocol portable/independent of the physical network, the packagers decided to make the maximum length of the IP datagram equal to the largest Maximum Transfer Unit (MTU) defined so far. However for other physical networks we must divide the datagrams to make it possible to pass through these networks. This is called fragmentation.

When a datagram is fragmented, each fragmented has its own header. A fragmented datagram may itself be fragmented if it encounters a network with an even smaller MTU. In another words, a datagram may be fragmented several times before it reached the final destination and also, the datagrams referred to as (frames in Data link layer) may arrives out of order at destination. Hence sorting of frames need to be done at the destination to recover the original data.

Output:

Enter The messgae to be Transmitted :
hi, it was nice meeting u on sunday

The Message has been divided as follows

Packet No.	Data
1	hi,
2	it
3	wa
4	s n
5	ice
6	me
7	eti
8	ng
9	u o
10	n s

11 und
12 ay

Packets received in the following order
4 2 6 3 5 1 8 9 11 7 12 10

Packets in order after sorting..
1 2 3 4 5 6 7 8 9 10 11 12

Message received is :
hi, it was nice meeting u on sunday

7. Problem:
Write a program for distance vector algorithm to find suitable path for transmission.

PROGRAM DESCRIPTION:

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 nonadaptive. Nonadaptive 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 (e.g., every ΔT sec, when the load changes, or when the topology changes), 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 neighbors.

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 DECnet and Novell's 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 out going 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 hat the receiver just time stamps and sends back as fast as possible.

The Count to Infinity Problem.

Distance vector routing algorithm reacts rapidly to good news, but leisurely to bad news. Consider a router whose best route to destination *X* is large. If on the next exchange neighbor *A* suddenly reports a short delay to *X*, the router just switches over to using the line to *A* to send traffic to *X*. In one vector exchange, the good news is processed.

To see how fast good news propagates, consider the five node (linear) subnet of following figure, where the delay metric is the number of hops. Suppose A is down initially and all the other routers know this. In other words, they have all recorded the delay to A as infinity.

A	B	C	D	E		A	B	C	D	E	
∞	∞	∞	∞	∞	Initially	1	2	3	4	Initially
1	∞	∞	∞	After 1 exchange	3	2	3	4	After 1 exchange
1	2	∞	∞	After 2 exchange	3	3	3	4	After 2 exchange
1	2	3	∞	After 3 exchange	5	3	5	4	After 3 exchange
1	2	3	4	After 4 exchange	5	6	5	6	After 4 exchange
					7	6	7	6	After 5 exchange
					7	8	7	8	After 6 exchange
						:			
					∞	∞	∞	∞	

Many ad hoc solutions to the count to infinity problem have been proposed in the literature, each one more complicated and less useful than the one before it. The **split horizon** algorithm works the same way as distance vector routing, except that the distance to *X* is not reported on line that packets for *X* are sent on (actually, it is reported as infinity). In the initial state of right figure, for example, *C* tells *D* the truth about distance to *A* but *C* tells *B* that its distance to *A* is infinite. Similarly, *D* tells the truth to *E* but lies to *C*.

Output

```
Enter the number the routers: 5
Enter 1 if the corresponding is adjacent to router A else enter 99:
        B C D E
Enter matrix:1 1 99 99
Enter 1 if the corresponding is adjacent to router B else enter 99:
        A C D E
Enter matrix:1 99 99 99
Enter 1 if the corresponding is adjacent to router C else enter 99:
        A B D E
Enter matrix:1 99 1 1
Enter 1 if the corresponding is adjacent to router D else enter 99:
        A B C E
Enter matrix:99 99 1 99
Enter 1 if the corresponding is adjacent to router E else enter 99:
        A B C D
Enter matrix:99 99 1 99

Router Table entries for router A
Destination Router: A B C D E
```

Outgoing Line: A B C C C
Hop Count: 0 1 1 2 2
Router Table entries for router B
Destination Router: A B C D E
Outgoing Line: A B A A A
Hop Count: 1 0 2 3 3
Router Table entries for router C
Destination Router: A B C D E
Outgoing Line: A A C D E
Hop Count: 1 2 0 1 1
Router Table entries for router D
Destination Router: A B C D E
Outgoing Line: C C C D C
Hop Count: 2 3 1 0 2
Router Table entries for router E
Destination Router: A B C D E
Outgoing Line: C C C C E
Hop Count: 2 3 1 2 0

8. Problem:
Write a Program in Java to implement TCP Client -Server architecture program.

9. Problem:

Write a Program in Java to implement UDP Client -Server architecture program.

10. Problem:
Write a program for simple RSA algorithm to encrypt and decrypt the data.

PROGRAM DESCRIPTION:
Theory

Cryptography has a long and colorful 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 ciphertext, is then transmitted, often by messenger or radio. The enemy, or intruder, hears and accurately copies down the complete ciphertext. However, unlike the intended recipient, he does not know the decryption key and so cannot decrypt the ciphertext 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 ciphertext. 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.

Algorithm

1. Generate two large random primes, P and Q, of approximately equal size.
2. Compute $N = P \times Q$
3. Compute $Z = (P-1) \times (Q-1)$.
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 $E \times D \equiv 1 \pmod{Z}$
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. $N = P \times Q = 11 \times 3 = 33$
   $Z = (P-1) \times (Q-1) = 10 \times 2 = 20$
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 $E \times D \equiv 1 \pmod{Z}$
   compute $D = E^{\wedge -1} \bmod Z = 3^{\wedge -1} \bmod 20$
   find a value for D such that Z divides $((E \times D)-1)$
   find D such that 20 divides 3D-1.
   Simple testing (D = 1, 2, ...) gives D = 7
   Check: $(E \times D)-1 = 3.7 - 1 = 20$, 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 = $M^{\wedge E}$ mod N
= $7^{\wedge 3}$ mod 33
= 343 mod 33
= 13.
Hence the ciphertext c = 13.

To check decryption we compute Message' = $C^{\wedge D}$ mod N
= $13^{\wedge 7}$ 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.

11. Problem:
Write a code how to create the socket an show the flow of the packets using TCP/IP protocols.