Experiment: 1

Objective:	Write a program to implement Lamport's logical clock.

Type of Algorithm: The algorithm of **Lamport's Logical Clock** is a simple algorithm used to determine the order of events in a distributed computer system.

Algorithm:

Lamport's logical clocks

- the "time" concept in distributed systems -- used to order events in a distributed system.
- assumption:
  - the execution of a process is characterized by a sequence of events. An event can be the execution of one instruction or of one procedure.
  - sending a message is one event, receiving a message is one event.
- The events in a distributed system are not total chaos. Under some conditions, it is possible to ascertain the order of the events. Lamport's logical clocks try to catch this.

Lamport's ``happened before'' relation

The ``happened before" relation ($\rightarrow$) is defined as follows:

- $A \rightarrow B$ if A and$B$ are within the same process (same sequential thread of control) and A occurred before B.
- if $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$

Event *Acausally affects* event *B*iff$A \rightarrow B$.

Distinct events A and B are *concurrent* ($A \mid\mid B$) if we do not have $A \rightarrow B$ or $B \rightarrow A$.

Lamport Logical Clocks

- are local to each process (processor?)
- do not measure real time
- only measure ``events"
- are consistent with the happened-before relation
- are useful for totally ordering transactions, by using logical clock values as timestamps

Logical Clock Conditions

C_i is the local clock for process P_i

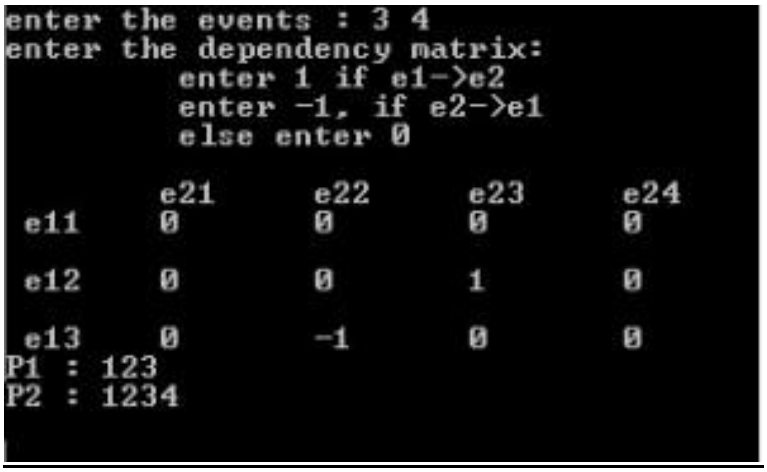
- if a and b are two successive events in P_i, then
  $C_i(b) = C_i(a) + d_1$, where $d_1 > 0$
- if a is the sending of message m by P_i, then m is assigned timestamp $t_m = C_i(a)$
- if b is the receipt of m by P_j, then
  $C_j(b) = \max\{C_j(b), t_m + d_2\}$, where $d_2 > 0$

The value of d could be 1, or it could be an approximation to the elapsed real time. For example, we could take d_1 to be the elapsed local time, and d_2 to be the estimated message transmission time. The latter solves the problem of waiting forever for a virtual time instant to pass.

Program Code:

```
#include<stdio.h>
#include<conio.h>
int max1(int a, int b)    //to find the maximum timestamp between two events
{
if (a>b)
return a;
else
return b;
}
int main()
{
int i,j,k,p1[20],p2[20],e1,e2,dep[20][20];
printf("enter the events : ");
scanf("%d %d",&e1,&e2);
for(i=0;i<e1;i++)
p1[i]=i+1;
for(i=0;i<e2;i++)
p2[i]=i+1;
printf("enter the dependency matrix:\n");
printf("\t enter 1 if e1->e2 \n\t enter -1, if e2->e1 \n\t else enter 0 \n\n");
for(i=0;i<e2;i++)
printf("\te2%d",i+1);
for(i=0;i<e1;i++)
{
printf("\n e1%d \t",i+1);
for(j=0;j<e2;j++)
scanf("%d",&dep[i][j]);
}

for(i=0;i<e1;i++)
{
for(j=0;j<e2;j++)
{
if(dep[i][j]==1)     //change the timestamp if dependency exist
{       p2[j]=max1(p2[j],p1[i]+1);
for(k=j;k<e2;k++)
p2[k+1]=p2[k]+1;
}
if(dep[i][j]==-1)    //change the timestamp if dependency exist
{
p1[i]=max1(p1[i],p2[j]+1);
for(k=i;k<e1;k++)
p2[k+1]=p1[k]+1;
}

}
}
printf("P1 : ");     //to print the outcome of Lamport Logical Clock
for(i=0;i<e1;i++)
{
printf("%d",p1[i]);
}
printf("\n P2 : ");
for(j=0;j<e2;j++)
printf("%d",p2[j]);

getch();
return 0 ;
}
```

Output 1:


enter the events : 3 4
enter the dependency matrix:
enter 1 if e1->e2
enter -1, if e2->e1
else enter 0
e21 e22 e23 e24
e11 0 0 0 0
e12 0 0 1 0
e13 0 -1 0 0
P1 : 123
P2 : 1234

Experiment: 2

Objective:	Write a program to implement Huang's Algorithm.

Type of Algorithm: **Huang's algorithm** is an algorithm for detecting termination in a distributed system.

Algorithm:

Huang's algorithm can be described by the following:

- Initially all processes are idle.
- A distributed task is started by a process sending a computational message to another process. This initial process which send the message is called the "controlling agent".
  - The initial weight of the controlling agent is w(usually 1).
- The following rules are applied throughout the computation:
  - A process sending a message splits its current weight between itself and the message.
  - A process receiving a message adds the weight of the message to itself.
  - Upon becoming idle, a process sends a message containing its entire weight back to the controlling agent and it goes idle.
  - Termination occurs when the controlling agent has a weight of $w$and is in the idle state.

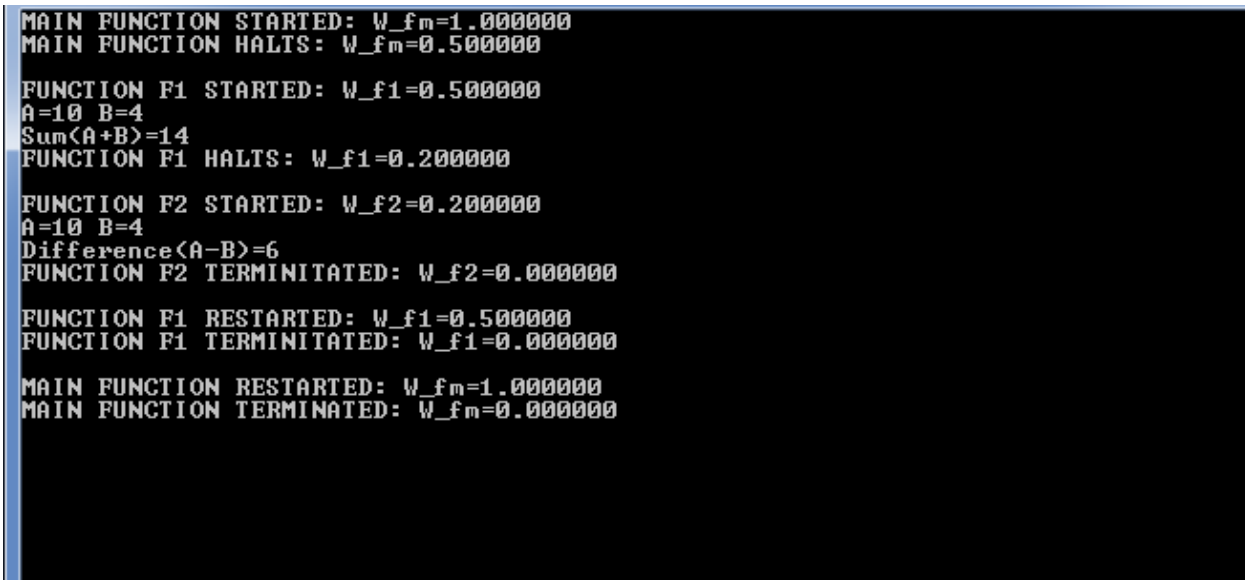
Some weaknesses to Huang's algorithm are that it is unable to detect termination if a message is lost in transit or if a process fails while in an active state.

Program Code:

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

float W_fm,W_f1,W_f2;

void f2()
{
        Int a,b;
        W_f2=0.2;
        printf("\n\nFUNCTION F2 STARTED: W_f2=%f",W_f2);
        a=10; b=4;
        printf("\nA=%d B=%d\nDifference(A-B)=%d",a,b,a-b);
        W_f2=0.0;
        printf("\nFUNCTION F2 TERMINITATED: W_f2=%f",W_f2);
}

void f1()
{
        Int a,b;
        Float W_f1=0.5;
        printf("\n\nFUNCTION F1 STARTED: W_f1=%f",W_f1);
        a=10; b=4;
        printf("\nA=%d B=%d\nSum(A+B)=%d",a,b,a+b);
        float W_f1=0.2;
        printf("\nFUNCTION F1 HALTS: W_f1=%f",W_f1);
```

```
        f2();
        W_f1=0.5;
        printf("\n\nFUNCTION F1 RESTARTED: W_f1=%f",W_f1);
        W_f1=0.0;
        printf("\nFUNCTION F1 TERMINITATED: W_f1=%f",W_f1);
}

int main(){
        float W_fm=1;
        printf("MAIN FUNCTION STARTED: W_fm=%f ",W_fm);
        float W_fm=0.5;
        printf("\nMAIN FUNCTION HALTS: W_fm=%f",W_fm);
        f1();
        W_fm=1;
        printf("\n\nMAIN FUNCTION RESTARTED: W_fm=%f",W_fm);
        W_fm=0;

        getch();
       }
```

Output : 2



Experiment: 3

Objective: Implement data transfer between client-server using socket programming.

Program Code:

Java Socket Server

```
package com.journaldev.socket;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.ClassNotFoundException;
import java.net.ServerSocket;
import java.net.Socket;

public class SocketServerExample {

   //static ServerSocket variable
   private static ServerSocket server;
   //socket server port on which it will listen
   private static int port = 9876;

   public static void main(String args[]) throws IOException, ClassNotFoundException{
      //create the socket server object
      server = new ServerSocket(port);
      //keep listens indefinitely until receives 'exit' call or program terminates
      while(true){
         System.out.println("Waiting for the client request");
         //creating socket and waiting for client connection
         Socket socket = server.accept();
         //read from socket to ObjectInputStream object
         ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
         //convert ObjectInputStream object to String
         String message = (String) ois.readObject();
         System.out.println("Message Received: " + message);
         //create ObjectOutputStream object
         ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
         //write object to Socket
         oos.writeObject("Hi Client "+message);
         //close resources
         ois.close();
         oos.close();
         socket.close();
         //terminate the server if client sends exit request
         if(message.equalsIgnoreCase("exit")) break;
      }
      System.out.println("Shutting down Socket server!!");
      //close the ServerSocket object
      server.close();
   }

}
```

Java Socket Client

```
package com.journaldev.socket;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;

public class SocketClientExample {

public static void main(String[] args)
throws UnknownHostException, IOException, ClassNotFoundException,InterruptedException
{
InetAddress host = InetAddress.getLocalHost();
Socket socket = null;
ObjectOutputStream oos = null;
ObjectInputStream ois = null;
for(int i=0; i<5;i++){
//establish socket connection to server
socket = new Socket(host.getHostName(), 9876);
//write to socket using ObjectOutputStream
oos = new ObjectOutputStream(socket.getOutputStream());
System.out.println("Sending request to Socket Server");
if(i==4)oos.writeObject("exit");
else oos.writeObject(""+i);
//read the server response message
ois = new ObjectInputStream(socket.getInputStream());
String message = (String) ois.readObject();
System.out.println("Message: " + message);
//close resources
ois.close();
oos.close();
Thread.sleep(100);
}
}
}
```

Output: 3

Output of java socket server

```
Waiting for the client request
Message Received: 0
Waiting for the client request
Message Received: 1
Waiting for the client request
Message Received: 2
Waiting for the client request
Message Received: 3
Waiting for the client request
Message Received: exit
Shutting down Socket server!!
```

Output of Java socket client:

```
Sending request to Socket Server
Message: Hi Client 0
Sending request to Socket Server
Message: Hi Client 1
Sending request to Socket Server
Message: Hi Client 2
Sending request to Socket Server
Message: Hi Client 3
Sending request to Socket Server
Message: Hi Client exit
```

Experiment: 4

Objective:	Implementation of **Interactive convergence algorithm**. (Clock Synchronization algo)

Program Code:

Server Code

```
import java.io.*;
import java.net.*;
import java.util.*;
public class Server
{
    private static Socket socket;
    public static void main(String[] args)
   {
      try
      {
int port = 2411;
ServerSocketserverSocket = new ServerSocket(port);
System.out.println("Server Started and listening to the port 2411");
          while(true)
         {
            socket = serverSocket.accept();
           InputStream is = socket.getInputStream();
InputStreamReaderisr = new InputStreamReader(is);
BufferedReaderbr = new BufferedReader(isr);
            String number = br.readLine();
System.out.println("Message received from client is "+number);
            String returnMessage;
intnum = Integer.parseInt(number);
            if(num==1)
            {
               Date time = new Date();
OutputStreamos = socket.getOutputStream();
ObjectOutputStreamoutputStream = new ObjectOutputStream(os);
outputStream.writeObject(time);
outputStream.flush();
System.out.println("Current time in millisec is "+time.getTime());
System.out.println("Current time of system is "+time);
               }
            else
            {
           returnMessage = "Invalid Input" + "\n";
           OutputStreamos = socket.getOutputStream();
           OutputStreamWriterosw = new OutputStreamWriter(os);
           BufferedWriterbw = new BufferedWriter(osw);
           bw.write(returnMessage);
           System.out.println("Message sent to the client is "+returnMessage);

                    bw.flush();
          }
        }
```

```
      }
      catch (Exception e)
      {          e.printStackTrace();
      }
      finally
      {
         try
         {
socket.close();
          }
         catch(Exception e)
            {
e.printStackTrace();
            }
      }
   }
}
```

Client Code

```
import java.io.*;
import java.net.*;
import java.util.*;
public class Client {
   private static Socket socket;
    public static void main(String args[])
   {
      try
      {
         String host = "localhost";
int port = 2411;
InetAddress address = InetAddress.getByName(host);
         socket = new Socket(address, port);
OutputStreamos = socket.getOutputStream();
OutputStreamWriterosw = new OutputStreamWriter(os);
BufferedWriterbw = new BufferedWriter(osw);
         String number = "1";
         String sendMessage = number + "\n";
bw.write(sendMessage);
bw.flush();
System.out.println("Message sent to the server : "+sendMessage);

InputStream is = socket.getInputStream();
ObjectInputStreaminStream = new ObjectInputStream(is);
        Date time =(Date)inStream.readObject();
         Date current = new Date();
         long t1 = time.getTime();
long t2 = current.getTime();
         long t3 = t2+t1;
         t3 = t3/2;
System.out.println("time received from the server : " +t1);
System.out.println("Current time of client is     : " +t2);
System.out.println("Average time in millisec is   : " +t3);
time.setTime(t3);
System.out.println("So current time is : " +time);
      }
      catch (Exception exception)
```

```
    {
exception.printStackTrace();
    }
    finally
    {
      try
      {
socket.close();
      }
      catch(Exception e)
      {
e.printStackTrace();
      }
    }
  }
}
```

Output: 4

```
run:
Server Started and listening to the port 2411
```

Experiment: 5

Objective:	Implement Remote Method Invocation (RMI).

Program Code:

Interface

```
import java.rmi.Remote;
import java.rmi.RemoteException;

// Creating Remote interface for our application
public interface Hello extends Remote {
   void printMsg() throws RemoteException;
}
```

Class

```
// Implementing the remote interface
public class ImplExample implements Hello {

   // Implementing the interface method
   public void printMsg() {
      System.out.println("This is an example RMI program");
   }
}
```

RMI server program

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server extends ImplExample {
   public Server() {}
   public static void main(String args[]) {
      try {
         // Instantiating the implementation class
         ImplExample obj = new ImplExample();

         // Exporting the object of implementation class
         // (here we are exporting the remote object to the stub)
         Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);

         // Binding the remote object (stub) in the registry
         Registry registry = LocateRegistry.getRegistry();

         registry.bind("Hello", stub);
         System.err.println("Server ready");
      } catch (Exception e) {
         System.err.println("Server exception: " + e.toString());
         e.printStackTrace();
      }
   }
}
```

RMI client program

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
   private Client() {}
   public static void main(String[] args) {
      try {
         // Getting the registry
         Registry registry = LocateRegistry.getRegistry(null);

         // Looking up the registry for the remote object
         Hello stub = (Hello) registry.lookup("Hello");

         // Calling the remote method using the obtained object
         stub.printMsg();

         // System.out.println("Remote method invoked");
      } catch (Exception e) {
         System.err.println("Client exception: " + e.toString());
         e.printStackTrace();
      }
   }
}
```

Output: 5

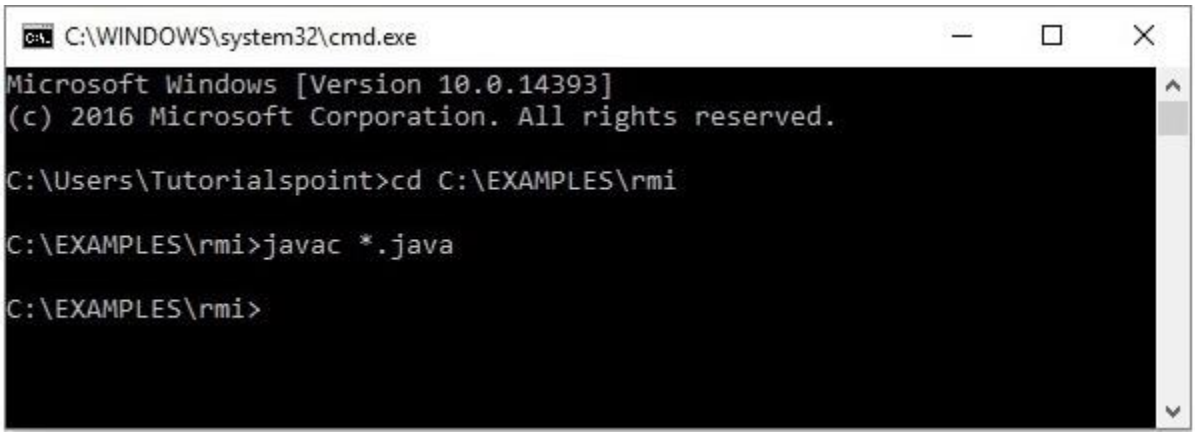
Compiling the Application

To compile the application −

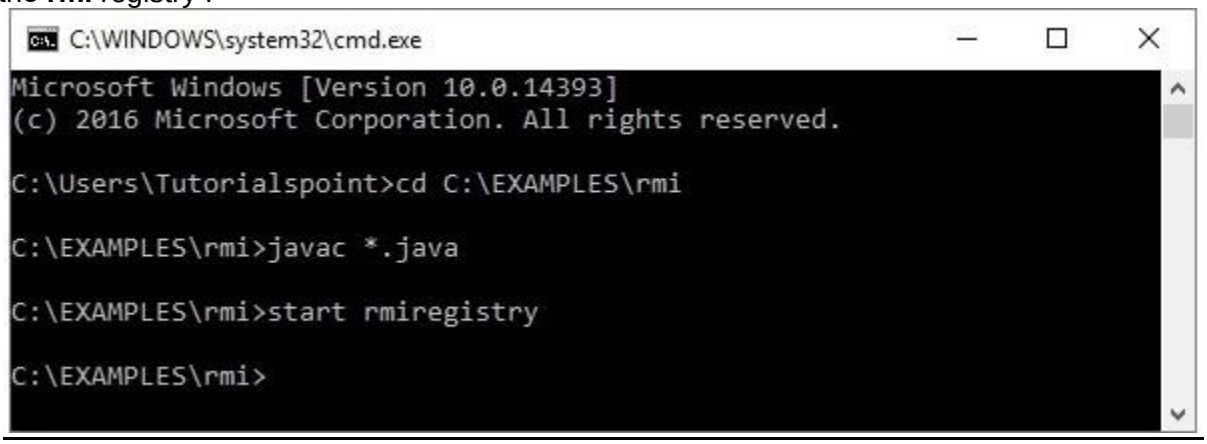
- Compile the Remote interface.
- Compile the implementation class.
- Compile the server program.
- Compile the client program.

Or,

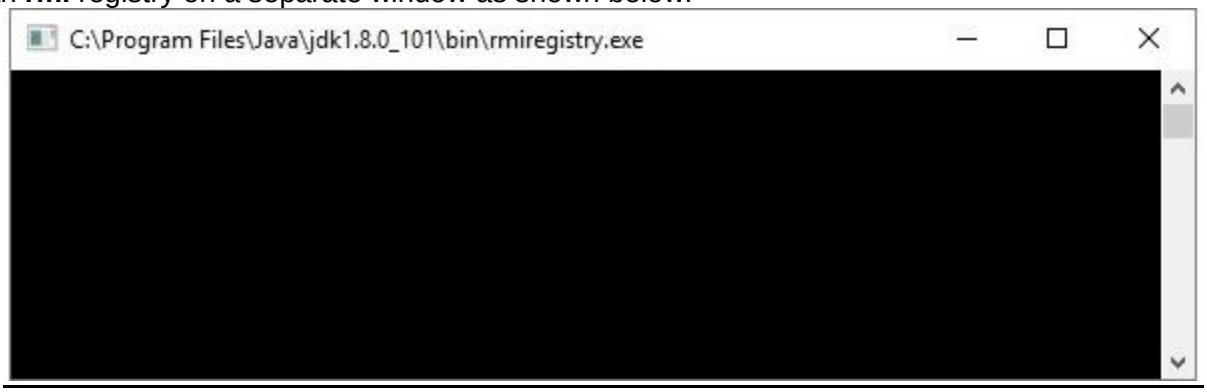
Open the folder where you have stored all the programs and compile all the Java files as shown below.



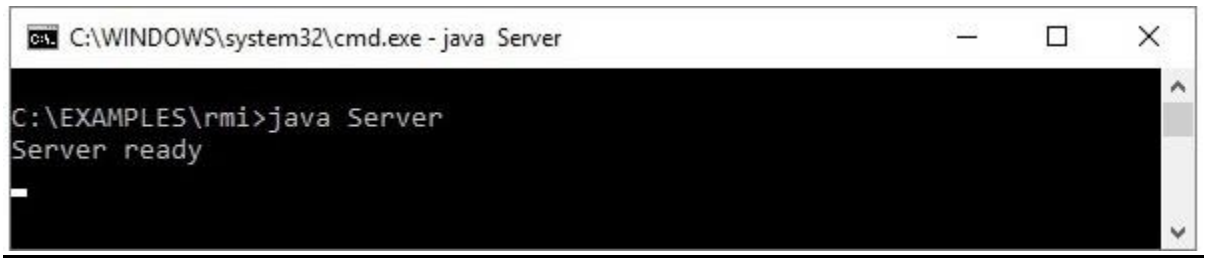
Step 1 − Start the **rmi** registry .



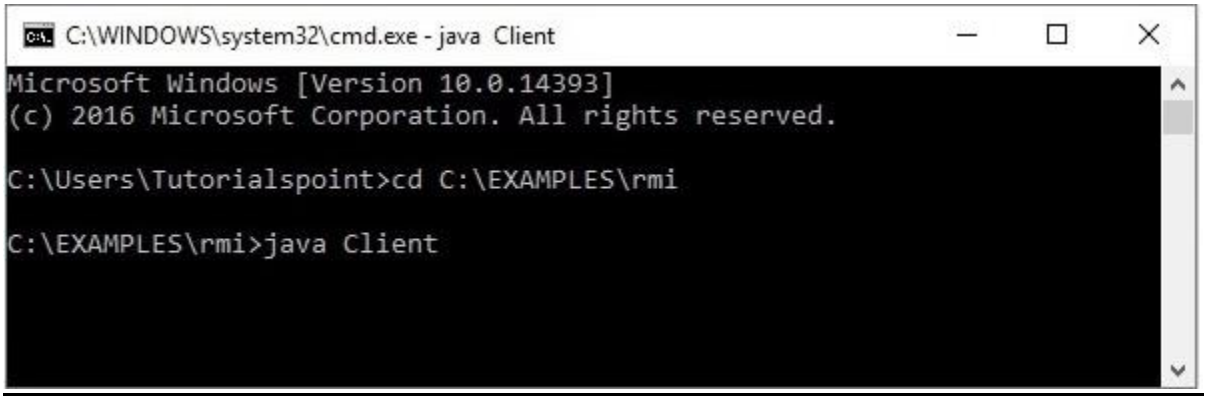
This will start an **rmi** registry on a separate window as shown below.



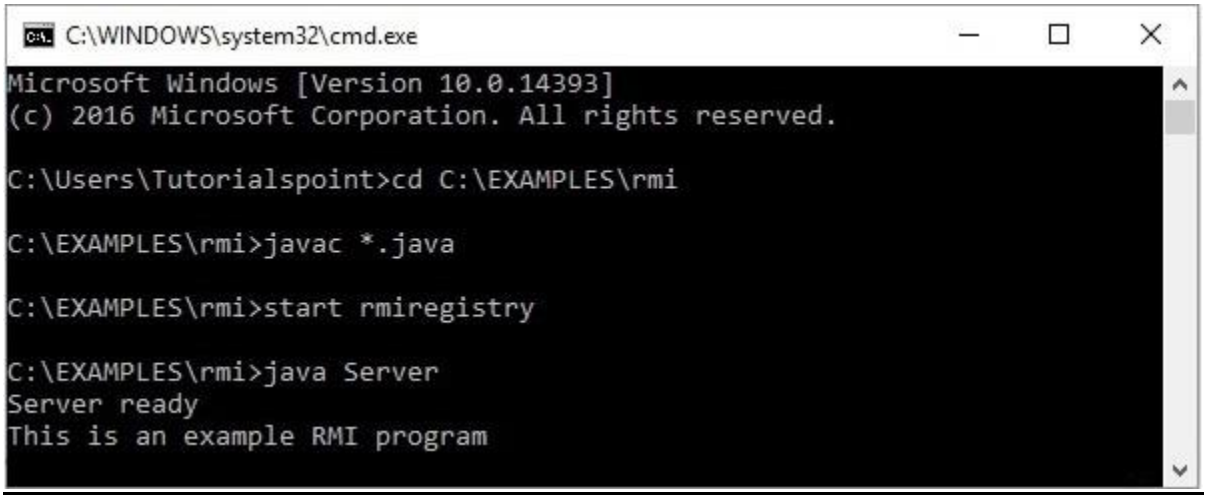
Step 2 − Run the server class file as shown below.



Step 3 − Run the client class file as shown below.



Verification − As soon you start the client, you would see the following output in the server.



Experiment: 6

Objective:	Study of CODA(Constant Data Availability) file system.

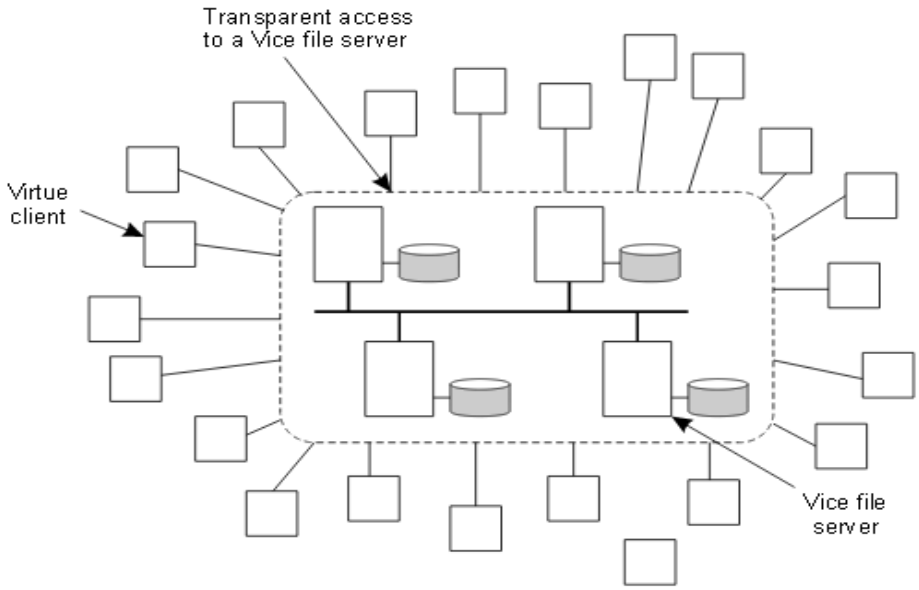
Coda is based on the Andrew File System (AFS).

Goals: naming and location transparency and high availability.

Feature:

- Disconnected operation for mobile computing.
- Is freely **available** under the GPL.
- **High** performance through client side persistent caching.
- Server replication.
- Security model for authentication, encryption and access control.
- Continued operation during partial network failures in server network.

Current activities on CODA



The overall organization of AFS

Communication:

Inter process communication in Coda is performed using RPCs. RPC2system for Coda is much more sophisticated than traditional RPC.

Process:

Coda uses a local cache to provide access to server **data** when the network connection is lost. During normal operation, a user reads and writes to the **file system** normally, while the client fetches, or "hoards", all of the **data** the user has listed as important in the event of network disconnection.

Naming & Location:

- The name space in Coda is hieratically structured as in UNIX and is partitioned into disjoint volumes.
- A VOLUME consists of a set of files & directories located on one server, & is the unit of replication in coda.
- Each files & directory is defined by a 96 bit long unique File Identifier (FID), Replicas of a file have same FID.

It corresponds to a partial sub tree in the shared name space as maintained by the Vice servers.

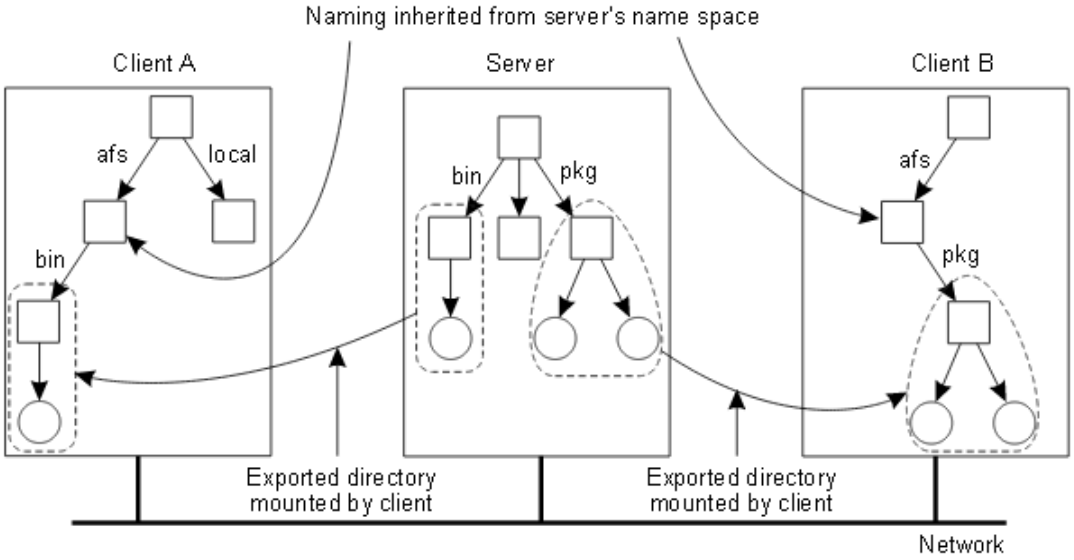


Fig: Client in Coda have access to a single shared name space

File Identifiers:

Considering that the collection of shared files may be replicated and distributed across multiple Vice servers, it becomes important to uniquely identify each file in such a way that it can be tracked to its physical location, while at the same time maintaining replication and location transparency.

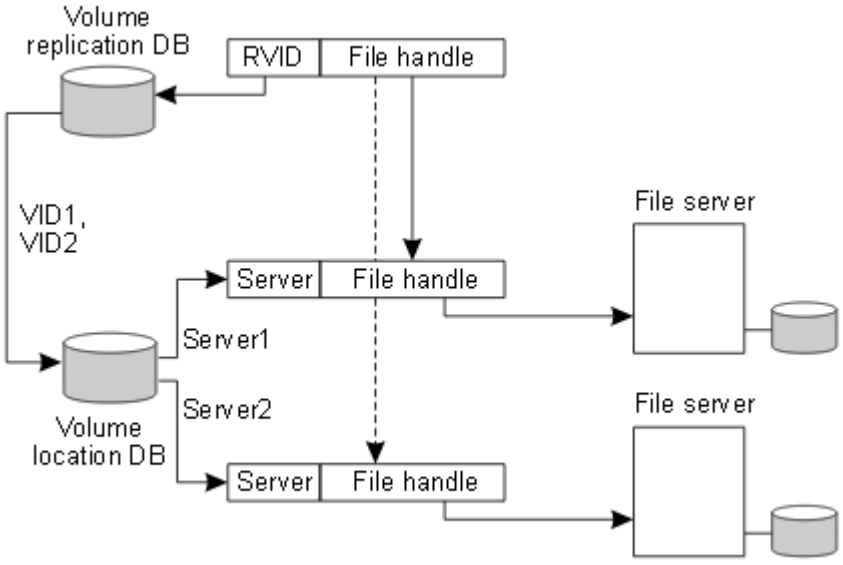


Fig: The implementation & resolution of a Coda file identifier

Experiment: 7

Objective:	Write a program to simulate the Mutual Exclusion in 'C'.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<time.h>

void main()
{
	int cs=0,pro=0;
	double run=5;
	char key='a';
	time_t t1,t2;

	clrscr();
	printf("Press a key(except q) to enter a process into critical section.");
	printf(" \nPress q at any time to exit.");

	t1 = time(NULL) - 5;
	while(key!='q')
	{
		while(!kbhit())
			if(cs!=0)
			{
				t2 = time(NULL);
				if(t2-t1 > run)
				{
					printf("Process%d ",pro-1);
    printf(" exits critical section.\n");
					cs=0;
				}
			}

		key = getch();
		if(key!='q')
		{
			if(cs!=0)
				printf("Error: Another process is currently executing critical section
Please wait till its execution is over.\n");
			else
			{
			 printf("Process %d ",pro);
				printf(" entered critical section\n");
				cs=1;
				pro++;
				t1 = time(NULL);
			}
		}
	}
}
```

Output:7

Press a key(except q) to enter a process into critical section.
Press q at any time to exit.

Process 0 entered critical section.

Error: Another process is currently executing critical section.
Please wait till its execution is over.

Process 0 exits critical section.

Process 1 entered critical section.

Process 1 exits critical section.

Process 2 entered critical section.

Error: Another process is currently executing critical section.
Please wait till its execution is over.

Process 2 exits critical section.

Experiment: 8

Objective:	Implement RPC mechanism for a file transfer across a network in 'C'

Program:

Server Code

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

#define IP_PROTOCOL 0
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0
#define nofile "File Not Found!"

// function to clear buffer
void  clearBuf(char* b)
{
   int  i;
   for  (i = 0; i < NET_BUF_SIZE; i++)
      b[i] = '\0';
}

// function to encrypt
char  Cipher(char  ch)
{
   return  ch ^ cipherKey;
}

// function sending file
int  sendFile(FILE* fp, char* buf, int  s)
{
   int  i, len;
   if  (fp == NULL) {
      strcpy(buf, nofile);
      len = strlen(nofile);
      buf[len] = EOF;
      for  (i = 0; i <= len; i++)
         buf[i] = Cipher(buf[i]);
      return  1;
```

```
    }

    char  ch, ch2;
    for  (i = 0; i < s; i++) {
        ch = fgetc(fp);
        ch2 = Cipher(ch);
        buf[i] = ch2;
        if  (ch == EOF)
            return  1;
    }
    return  0;
}

// driver code
int  main()
{
    int  sockfd, nBytes;
    struct  sockaddr_in addr_con;
    int  addrlen = sizeof(addr_con);
    addr_con.sin_family = AF_INET;
    addr_con.sin_port = htons(PORT_NO);
    addr_con.sin_addr.s_addr = INADDR_ANY;
    char  net_buf[NET_BUF_SIZE];
    FILE* fp;

    // socket()
    sockfd = socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);

    if  (sockfd < 0)
        printf("\nfile descriptor not received!!\n");
    else
        printf("\nfile descriptor %d received\n", sockfd);

    // bind()
    if  (bind(sockfd, (struct  sockaddr*)&addr_con, sizeof(addr_con)) == 0)
        printf("\nSuccessfully binded!\n");
    else
        printf("\nBinding Failed!\n");

    while  (1) {
        printf("\nWaiting for file name...\n");

        // receive file name
        clearBuf(net_buf);

        nBytes = recvfrom(sockfd, net_buf,
                          NET_BUF_SIZE, sendrecvflag,
                          (struct  sockaddr*)&addr_con, &addrlen);

        fp = fopen(net_buf, "r");
        printf("\nFile Name Received: %s\n", net_buf);
        if  (fp == NULL)
            printf("\nFile open failed!\n");
        else
            printf("\nFile Successfully opened!\n");

        while  (1) {
```

```
        // process
        if  (sendFile(fp, net_buf, NET_BUF_SIZE)) {
           sendto(sockfd, net_buf, NET_BUF_SIZE,
                 sendrecvflag,
              (struct  sockaddr*)&addr_con, addrlen);
           break;
        }

        // send
        sendto(sockfd, net_buf, NET_BUF_SIZE,
              sendrecvflag,
           (struct  sockaddr*)&addr_con, addrlen);
        clearBuf(net_buf);
     }
     if  (fp != NULL)
        fclose(fp);
  }
  return  0;
}
```

Client Code

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

#define IP_PROTOCOL 0
#define IP_ADDRESS "127.0.0.1" // localhost
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0

// function to clear buffer
void  clearBuf(char* b)
{
  int  i;
  for  (i = 0; i < NET_BUF_SIZE; i++)
     b[i] = '\0';
}

// function for decryption
char  Cipher(char  ch)
{
  return  ch ^ cipherKey;
}

// function to receive file
int  recvFile(char* buf, int  s)
{
  int  i;
  char  ch;
```

```
   for  (i = 0; i < s; i++) {
      ch = buf[i];
      ch = Cipher(ch);
      if  (ch == EOF)
         return  1;
      else
         printf("%c", ch);
   }
   return  0;
}

// driver code
int  main()
{
   int  sockfd, nBytes;
   struct  sockaddr_in addr_con;
   int  addrlen = sizeof(addr_con);
   addr_con.sin_family = AF_INET;
   addr_con.sin_port = htons(PORT_NO);
   addr_con.sin_addr.s_addr = inet_addr(IP_ADDRESS);
   char  net_buf[NET_BUF_SIZE];
   FILE* fp;

   // socket()
   sockfd = socket(AF_INET, SOCK_DGRAM,
                IP_PROTOCOL);

   if  (sockfd < 0)
      printf("\nfile descriptor not received!!\n");
   else
      printf("\nfile descriptor %d received\n", sockfd);

   while  (1) {
      printf("\nPlease enter file name to receive:\n");
      scanf("%s", net_buf);
      sendto(sockfd, net_buf, NET_BUF_SIZE,
           sendrecvflag, (struct  sockaddr*)&addr_con,
           addrlen);

      printf("\n---------Data Received---------\n");

      while  (1) {
         // receive
         clearBuf(net_buf);
         nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,
                     sendrecvflag, (struct  sockaddr*)&addr_con,
                     &addrlen);

         // process
         if  (recvFile(net_buf, NET_BUF_SIZE)) {
            break;
         }
      }
      printf("\n-------------------------------\n");
   }
   return  0;
}
```

Output: 8

Server

```
Socket file descriptor 3 received

Successfully binded!

Waiting for file name...

File Name Received: dm.txt

File Successfully opened!

Waiting for file name...

File Name Received: /home/dmayank/Documents/dm.txt

File Successfully opened!
```

Client

```
Socket file descriptor 3 received

Please enter file name to receive:
dm.txt

---------Data Received---------
30
-------------------------------

Please enter file name to receive:
/home/dmayank/Documents/dm.txt

---------Data Received---------
30
-------------------------------
```

Index

S.No.	Experiment	Page No.	Date	Teacher Remark
1	Write a program to implement Lamport's logical clock.			
2	Write a program to implement Huang's Algorithm.			
3	Implement data transfer between client-server using socket programming.			
4	Implementation of Interactive convergence algorithm. (Clock Synchronization algo)			
5	Implement Remote Method Invocation (RMI).			
6	Study of CODA(Constant Data Availability) file system.			
7	Write a program to simulate the Mutual Exclusion in 'C'.			
8	Implement RPC mechanism for a file transfer across a network in 'C'			