

NUTAN MAHARASHTRA INSTITUTE OF ENGINEERING AND TECHNOLOGY

“Samarth Vidya Sankul” Talegaon Dabhade, PUNE-410507

**DEPARTMENT OF INFORMATION TECHNOLOGY
ENGINEERING
SEMESTER-II**



**Laboratory Practice V
(414454)**

LABORATORY MANUAL

Savitribai Phule Pune University		
Final Year of Information Technology Engineering (2019 Course)		
414454: Laboratory Practice V		
Teaching Scheme:	Credit	Examination Scheme:
PR: 04 Hours/Week	02	TW: 50 Marks PR: 25 Marks

Prerequisite Courses:

- Operating Systems
- Computer Network Technology
- Web Application Development

Course Objectives:

1. The course aims to provide an understanding of the principles on which the distributed systems are based, their architecture, algorithms and how they meet the demands of Distributed applications.
2. The course covers the building blocks for a study related to the design and the implementation of distributed systems and applications.

Course Outcomes:

On completion of the course, student will be able to-

Cloud Computing :

- | | |
|------|---|
| CO1: | Demonstrate knowledge of the core concepts and techniques in distributed systems. |
| CO2: | Learn how to apply principles of state-of-the-Art Distributed systems in practical application. |
| CO3: | Design, build and test application programs on distributed systems. |

List of Assignments

TITLE
Group A
<ol style="list-style-type: none">1. Implement multi-threaded client/server Process communication using RMI.2. Develop any distributed application using CORBA to demonstrate object brokering. (Calculator or String operations).3. Develop a distributed system, to find sum of N elements in an array by distributing N/n elements to n number of processors MPI or OpenMP. Demonstrate by displaying the intermediate sums calculated at different processors.4. Implement Berkeley algorithm for clock synchronization.5. Implement token ring based mutual exclusion algorithm Design and deploy a web application in a PaaS environment.6. Implement Bully and Ring algorithm for leader election.7. Create a simple web service and write any distributed application to consume the web service.8. Mini Project (In group): A Distributed Application for Interactive Multiplayer Games
CASE STUDIES
<ul style="list-style-type: none">• Client/Server Process communication• Create a simple web service and write any distributed application• Create Interactive Multiplayer Games

Assignment No: 1

AIM:

Implement multi-threaded client/server Process communication using RMI.

Objective:

Multi-threading.

How Remote Method Invocation Work.

How RMI allows objects to invoke methods on remote objects.

How to write Distributed Object Application.

Outcome:

Implement multi-threaded client/server Process communication using RMI.

Explanation:

Tools used:

1. Windows
2. Linux
3. Java language

RMI (Remote Method Invocation) is used for distributed object references system. A distributed object is an object which publishes its interface on other machines. A Remote Object is a distributed object whose state is encapsulated. Stub and Skeleton are two objects used to communicate with the remote object.

Stub: Stub is a gateway for client program which is used to communicate with skeleton object, by establishing a connection between them.

Skeleton: Resides on Server program which is used for passing the request from stub to the remote interface.

Steps to Run Java RMI Application

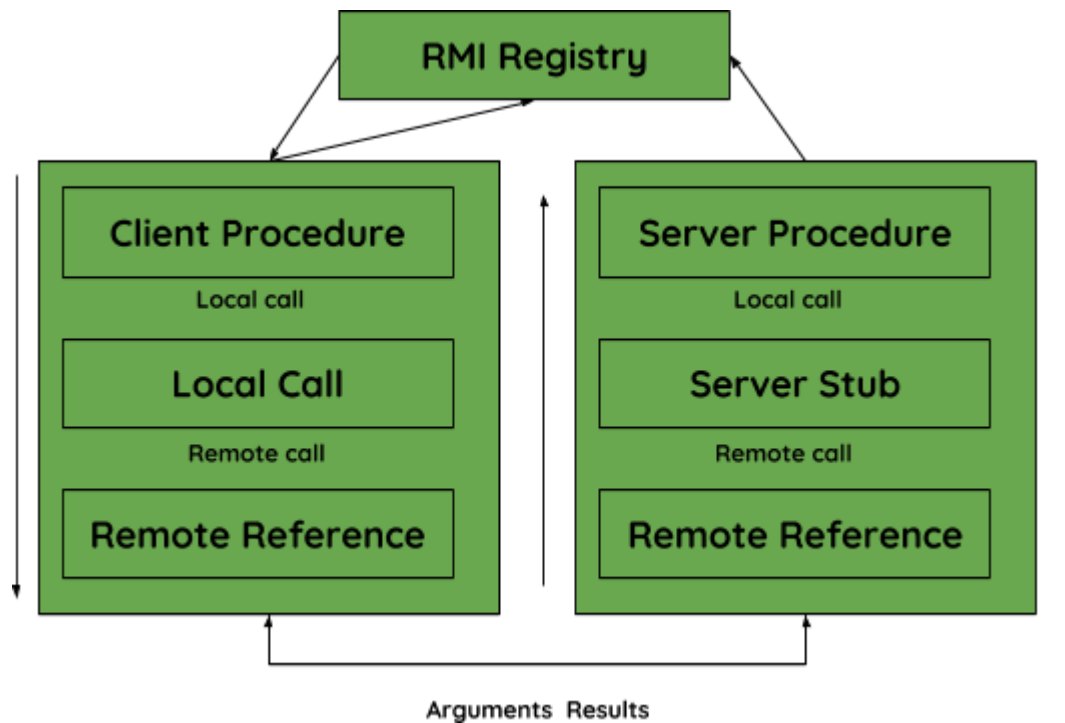
Creation of classes and interfaces for the problem statement:

The steps involved in this are as follows:

Problem Statement:

Create an RMI Application for finding the addition of a number

How communication and process take place in RMI:



1. Create the remote interface

```
import java.rmi.*;
public interface Adder extends Remote {
    public int add (int x, int y) throws RemoteException;
}
```

2. Provide the implementation of the remote interface

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
    AdderRemote()throws RemoteException{
        super();
    }
    public int add(int x, int y){return x+y;}
}
```

3. Create and run the server application

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
    public static void main(String args[]){
        try{
            Adder stub=new AdderRemote();
```

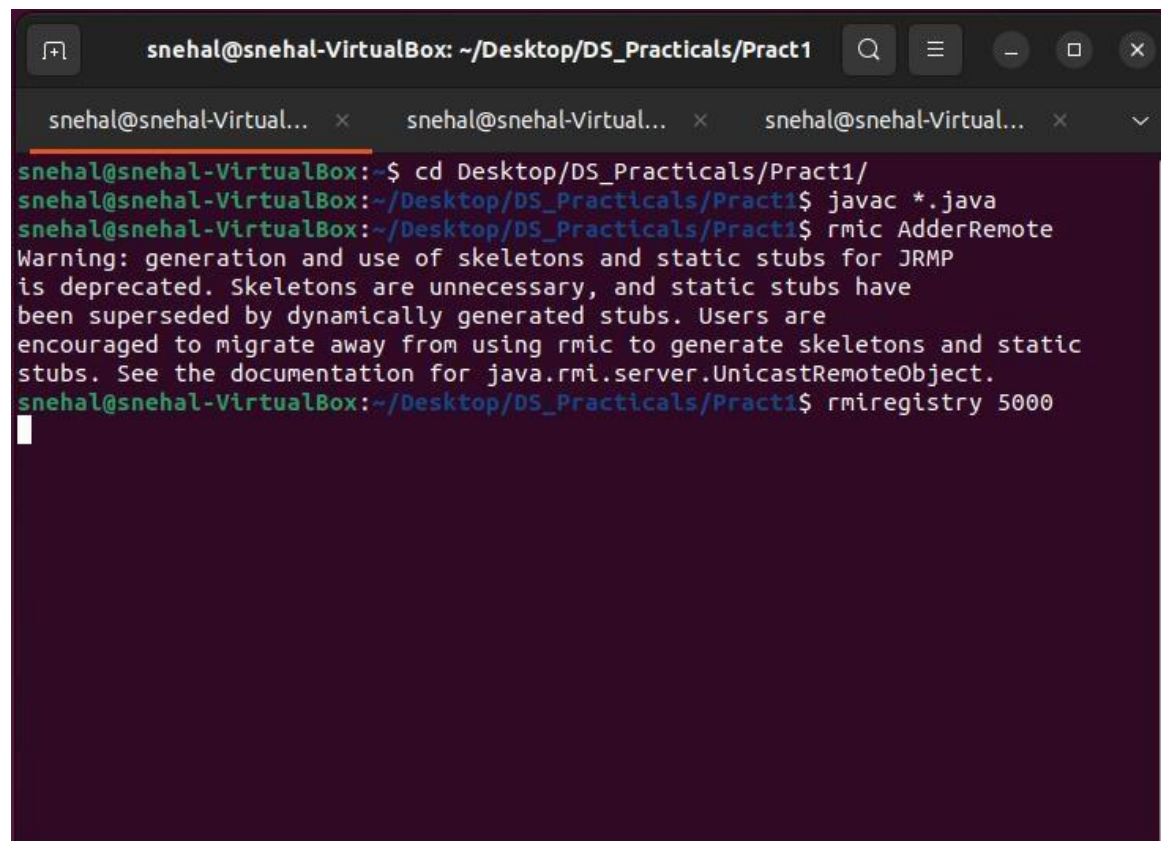
```
Naming.rebind("rmi://localhost:5000/sonoo",stub);  
} catch(Exception e){System.out.println(e);}  
}  
}
```

4. Create and run the client application

```
import java.rmi.*;  
public class MyClient{  
    public static void main(String args[]){  
        try{  
            Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");  
            System.out.println(stub.add(34,4));  
        } catch(Exception e){}  
    }  
}
```

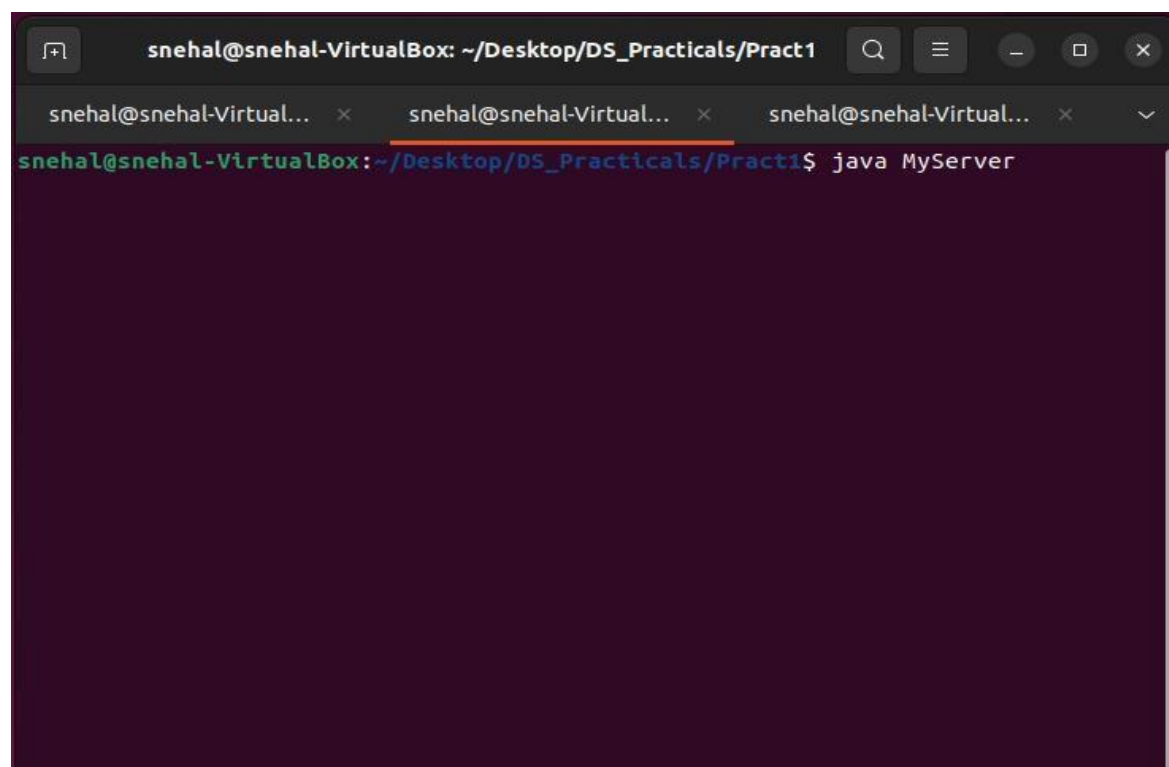
Output of the program:

- 1) compile all the java files**
javac *.java
- 2) create stub and skeleton object by rmic tool**
rmic AdderRemote
- 3) start rmi registry in one command prompt**
rmiregistry 5000
- 4) start the server in another command prompt**
java MyServer
- 5) start the client application in another command prompt**
java MyClient



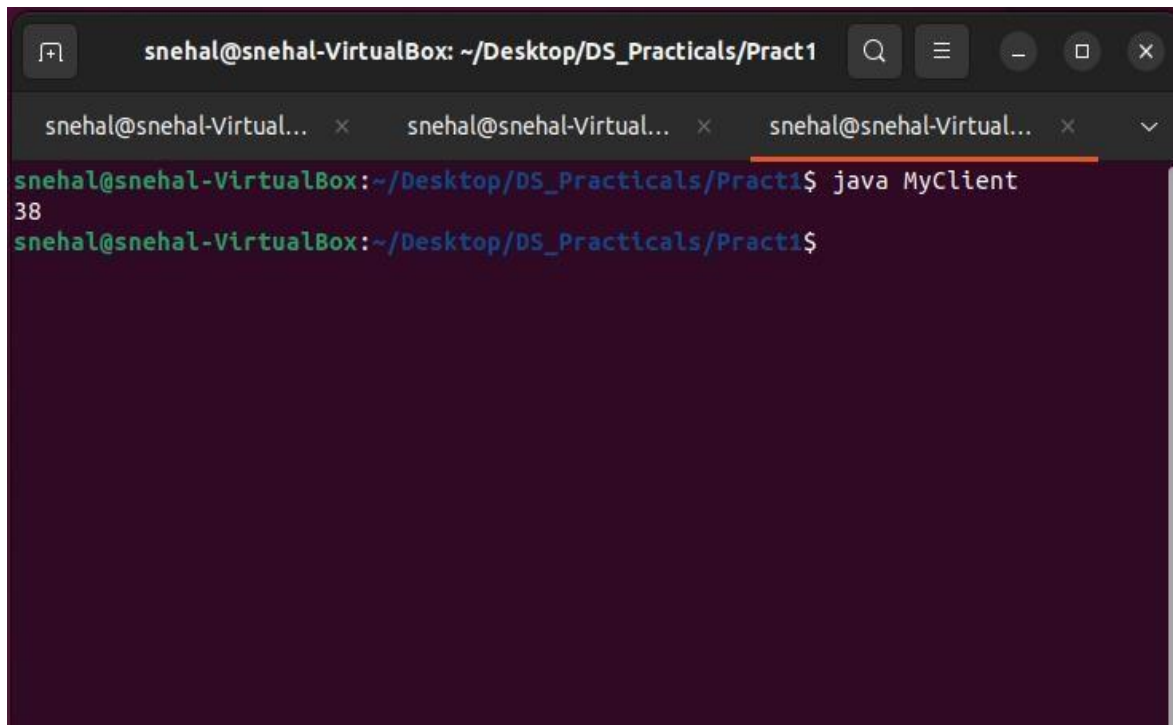
A terminal window titled 'snehal@snehal-VirtualBox: ~/Desktop/DS_Practicals/Pract1'. The terminal shows the following commands and output:

```
snehal@snehal-VirtualBox:~$ cd Desktop/DS_Practicals/Pract1/
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract1$ javac *.java
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract1$ rmic AdderRemote
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract1$ rmiregistry 5000
```



A terminal window titled 'snehal@snehal-VirtualBox: ~/Desktop/DS_Practicals/Pract1'. The terminal shows the following command and output:

```
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract1$ java MyServer
```



The screenshot shows a terminal window titled 'snehal@snehal-VirtualBox: ~/Desktop/DS_Practicals/Pract1'. The terminal has a dark background with green and blue text. The prompt is 'snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract1\$'. The user has entered the command 'java MyClient', and the output is '38'. The prompt is now 'snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract1\$'.

```
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract1$ java MyClient
38
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract1$
```

Result:

Successfully the Client-Server communication is done

Assignment No: 2

AIM:

Develop any distributed application using CORBA to demonstrate object brokering.
(Calculator or String operations).

Objective:

Basic implementation of a Java/CORBA application using static invocations.

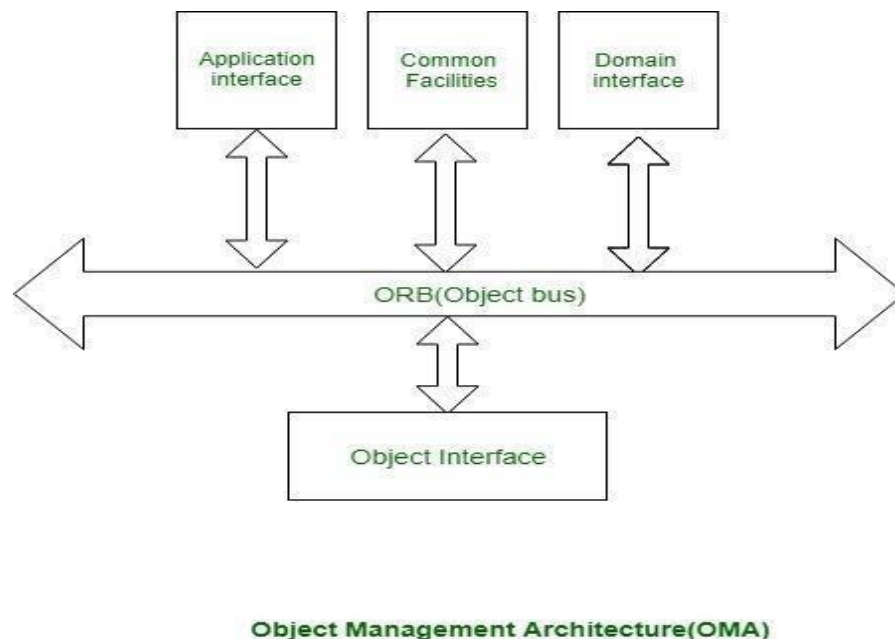
Outcome:

CORBA also provides a dynamic invocation capability where clients can discover the interfaces on-the-fly.

Explanation:

Common Object Request Broker Architecture (CORBA) could be a specification of a regular design for middleware. It is a client-server software development model
CORBA Reference Model:

The CORBA reference model known as Object Management design (OMA) is shown below figure. The OMA is itself a specification (actually, a group of connected specifications) that defines a broad vary of services for building distributed client-server applications



Divide and conquer. The remote objects can be independently designed.

Increase reusability. It is often possible to design the remote objects so that other systems can use them too.

Increase reuse. You may be able to reuse remote objects that others have created.

Design for flexibility. The broker objects can be updated as required, or you can redirect the proxy to communicate with a different remote object.

Design for portability. You can write clients for new platforms while still accessing brokers and remote objects on other platforms.

Design defensively. You can provide careful assertion checking in the remote objects.

The Calculator sample application is a basic implementation of a Java/CORBA application using static invocations. A static invocation is when a client is aware of what interfaces and methods are available at compile time. Note: CORBA also provides a dynamic invocation capability where clients can discover the interfaces on-the-fly.

The Java/CORBA development process for the Calculator example application is broken down in to ten manageable steps:

1. Download and install a Java ORB
2. Create IDL file
3. Compile IDL file
4. Create the client
5. Create the server
6. Create the interface implementation
7. Compile the client
8. Compile the server
9. Compile the interface implementation
10. Start the naming service (OS Agent)
11. Start the server
12. Start the client.

Program for Calculator Application

1. Calc.idl:

```
module WssCalculator
{
    interface Calc
    {
        //Performs the Calculations:ADD/SUB/MUL/DIV
        long calculate(in long operator,in long num1,in long num2);

        //The Server EXITS when the Client prompts it to do so
```

```
oneway void shutdown();  
};  
};
```

2. CalcServer.java:

```
//Importing all the packages and classes  
//Import the package which contains the Server Skeleton  
import WssCalculator.*;  
  
//Import the below two packages to use the Naming Service  
import org.omg.CosNaming.*;  
import org.omg.CosNaming.NamingContextPackage.*;  
  
//Import this package to run the CORBA Application  
import org.omg.CORBA.*;  
  
//Import the below to Classes for inheriting Portable Server  
import org.omg.PortableServer.*;  
import org.omg.PortableServer.POA;  
  
//Initiate the ORB using the class Properties  
import java.util.Properties;  
  
//Perform the Input-Output functionalities  
import java.io.*;  
import java.util.*;  
  
//Write the Servant class  
//It inherits the general CORBA utilities generated by the Compiler  
  
class Calcserverimpl extends CalcPOA  
{  
  
    //orb variable is used to invoke the shutdown()  
    private ORB orb;  
  
    public void setORB(ORB orb_val)  
    {  
        orb = orb_val;  
    }  
  
    //Declaring and Implementing the required method  
    public int calculate(int a,int b,int c)
```

```
{
//ADDITION
if(a==43)
{
return (b+c);
}

//SUBTRACTION
else if(a==45)
{
return (b-c);
}

//MULTIPLICATION
else if(a==42)
{
return (b*c);
}

//DIVISION
else if(a==47)
{
return (b/c);
}

//DEFAULT
else
{
return 0;
}
}

//Closing the server
public void shutdown()
{
orb.shutdown(false);
}
} //end of the servant class

public class CalcServer
{
public static void main(String args[])
{
```

```
try
{
//Create and Initialize the ORB object
//init() allows to set the properties at run time
ORB orb=ORB.init(args,null);

//Obtain the initial Naming Context
//Obtain an initial object reference to the name server

//orb retrieves the reference to the Root POA
//Activate the POA Manager
//activate() causes the POAs to process the client requests

POA rootpoa=POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
rootpoa.the_POAManager().activate();

//The server instantiates the servant objects
//The servant performs the operations defined in the idlj interface

Calcservimpl simpl=new Calcservimpl();
simpl.setORB(orb);

//Get the object reference associated with the servant
//narrow() is used to cast CORBA obj ref to its proper type

org.omg.CORBA.Object ref = rootpoa.servant_to_reference(simpl);
Calc href=CalcHelper.narrow(ref);

//Obtain the initial Naming Context
//Obtain an object reference to the Name Server
org.omg.CORBA.Object objRef=orb.resolve_initial_references("NameService");

//Narrow the objref to its proper type
NamingContextExt ncRef=NamingContextExtHelper.narrow(objRef);

//Register the Servant with the Name Server
String name = "Calc";

//NameComponent array contains the path to Calc
NameComponent path[]=ncRef.to_name(name);

//Pass the path and the servant object to the Naming Service
//Bind the servant object to Calc
```

```
ncRef.rebind(path,href);

System.out.println("TheER is READY");
System.out.println("TheER is WAITING to receive the CLIENT requests");

//run() is called by the main thread
//run() enables the ORB to perform work using the main thread
//the server waits until an invocation comes from the ORB

orb.run();
}

catch (Exception e)
{
System.err.println("ERROR e);
e.printStackTrace(System.out);
}

//This statement is executed when the Client wishes to discontinue
System.out.println("Theer Exits");
} //end of main()
} //end of CalcServer()
```

3. CalcClient.java:

```
//Import all the important packages

//Import the package which contains the Client Stub
import WssCalculator.*;

//Import the below two packages to use the Naming Service
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;

//Import this package to run the CORBA Application
import org.omg.CORBA.*;

//Import to perform Input-Output functionalities
import java.io.*;
import java.util.*;

public class CalcClient
{
static Calc cimpl;
```

```
public static void main(String args[])
{
try
{
//Declaring and initializing the variables
int dec=1;
int i=0;
int j=0;
int k=0;
int result=0;
int x=1;
char c='x';
char d='y';
char f='z';
String abc="vas";

//Create and Initialize the ORB object
//init() allows to set properties at run time

ORB orb=ORB.init(args,null);

//ORB helps the Client to locate the actual services which it needs
//COS Naming Service helps the client to do so

//Obtain the initial Naming Context
//Obtain an object reference to the name server
org.omg.CORBA.Object objRef=orb.resolve_initial_references("NameService");

//Narrow the objref to its proper type
NamingContextExt ncRef=NamingContextExtHelper.narrow(objRef);

//Identify a String to refer the Naming Service to Calc object
String name="Calc";

//Get a reference to the CalcServer and Narrow it to Calc object
cimpl=CalcHelper.narrow(ncRef.resolve_str(name));
System.out.println("Obtainedndle on the server object");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

while(x==1)
{
System.out.println("Enterstring:");
```

```
abc=br.readLine();

//Separate the input string into separate characters
c=abc.charAt(0);
d=abc.charAt(1);
f=abc.charAt(2);

//Get the ASCII value of the Operator
i=(int)c;

//Get the Integer values of the other two characters
j=Character.getNumericValue(d);
k=Character.getNumericValue(f);

result=cimpl.calculate(i,j,k);
System.out.println("The result of the operation is "+result);

System.out.println("Enter continue and 0 to exit ");
x=Integer.parseInt(br.readLine());
}

//If the Client wants to discontinue
cimpl.shutdown();
}

catch(Exception e)
{
System.out.println("ERROR+ e) ;
e.printStackTrace(System.out);
}
} //end of main()
} //end of class
```

4. Compile both the files by entering the following commands:

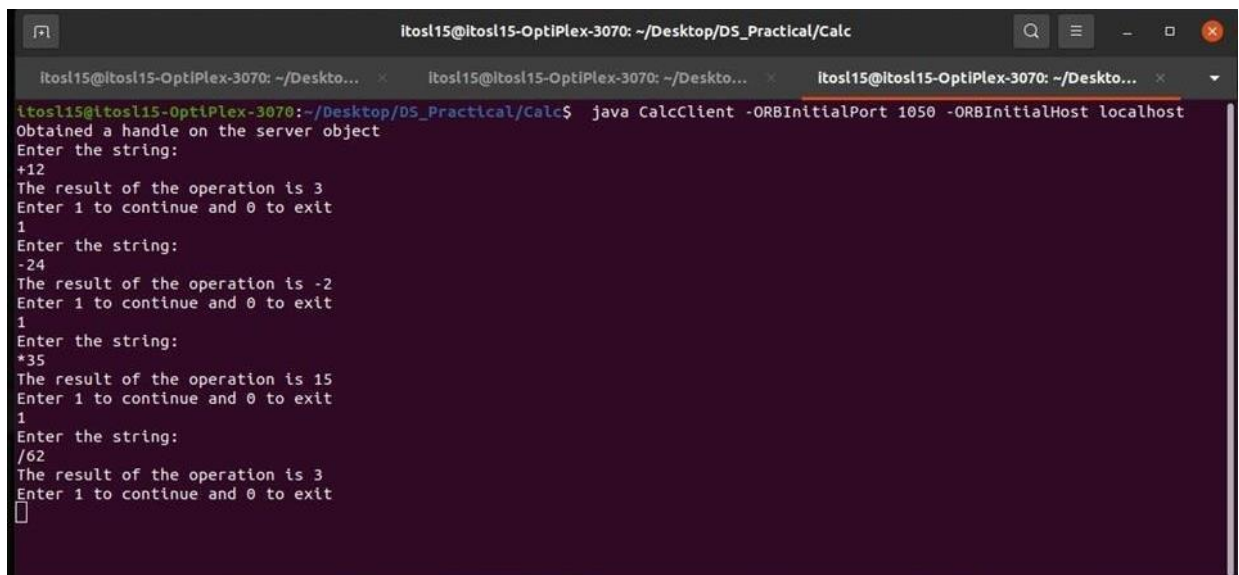
```
javac CalcServer.java WssCalculator/*.java
```

```
javac CalcClient.java WssCalculator/*.java
```


Output:

```
itosl15@itosl15-OptiPlex-3070: ~/Desktop/DS_Practical/Calc
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/Calc$ cd Desktop/DS_Practical/Calc/
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/Calc$ idlj -fall Calc.idl
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/Calc$ javac CalcServer.java WssCalculator/*.java
Note: WssCalculator/CalcPOA.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/Calc$
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/Calc$ javac CalcClient.java WssCalculator/*.java
Note: WssCalculator/CalcPOA.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/Calc$ orbd -ORBInitialPort 1050 orbd -ORBInitialHost localhost
□
```

```
itosl15@itosl15-OptiPlex-3070: ~/Desktop/DS_Practical/Calc
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/Calc$ java CalcServer -ORBInitialPort 1050 -ORBInitialHost localhost
The SERVER is READY
The SERVER is WAITING to receive the CLIENT requests
□
```



```
itosl15@itosl15-OptiPlex-3070: ~/Desktop/DS_Practical/Calc
itosl15@itosl15-OptiPlex-3070: ~/Desktop/DS_Practical/Calc$ java CalcClient -ORBInitialPort 1050 -ORBInitialHost localhost
Obtained a handle on the server object
Enter the string:
+12
The result of the operation is 3
Enter 1 to continue and 0 to exit
1
Enter the string:
-24
The result of the operation is -2
Enter 1 to continue and 0 to exit
1
Enter the string:
*35
The result of the operation is 15
Enter 1 to continue and 0 to exit
1
Enter the string:
/62
The result of the operation is 3
Enter 1 to continue and 0 to exit
1
```

Result:

Successfully Created CORBA system for Timestamp application.

Assignment No 3

AIM:

Develop a distributed system, to find sum of N elements in an array by distributing N/n elements to n number of processors MPI or OpenMP. Demonstrate by displaying the intermediate sums calculated at different processors

Objective:

To learn the Demonstration by displaying the intermediate sums calculated at different processors

Outcome:

Find sum of N elements in an array by distributing N/n elements.

Explanation:

Library of routines that can be used to create parallel programs in C or Fortran77. It allows users to build parallel applications by creating parallel processes and To reduce the time complexity of the program, parallel execution of sub-arrays is done by parallel processes running to calculate their partial sums and then finally, the master process (root process) calculates the sum of these partial sums to return to The network nodes communicate among themselves in order to decide which of them will get into the "coordinator" state. For that, they need some method in order to break the symmetry among them. For example, if each node has unique and comparable identities, then the nodes can compare their identities, and decide that the node with the highest identity is the coordinator total sum of the array. Exchange information among these processes.

Installation of OPENMPI

1. Download openmpi-4.1.4.tar.bz2 from <http://www.open-mpi.org> in a folder say LP5.
2. Goto the terminal (Command prompt)
3. update using

```
sudo apt-get update
sudo apt install gcc {if not already installed}
```
4. Goto the directory which contains the downloaded file
5. Extract the files using

```
tar -jxf openmpi-4.1.4.tar.bz2
```
6. The directory openmpi-4.1.4 is created
7. Configure, compile and install by executing the following commands

```
./configure --prefix=$HOME/opt/openmpi
make all
make install
```

8. Now openmpi folder is created in 'opt' folder of Home directory.
9. Now the folder LP5 can be deleted (optional)
10. Update the PATH and LD_LIBRARY_PATH environment variable using

```
echo "export PATH=$PATH:$HOME/opt/openmpi/bin" >> $HOME/.bashrc
echo"                                                                    export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/opt/openmpi/lib"
>> $HOME/.bashrc
```
11. Compile the program using

```
mpicc name of the program
```
12. Execute the program using

```
mpirun -np N ./a.out
```

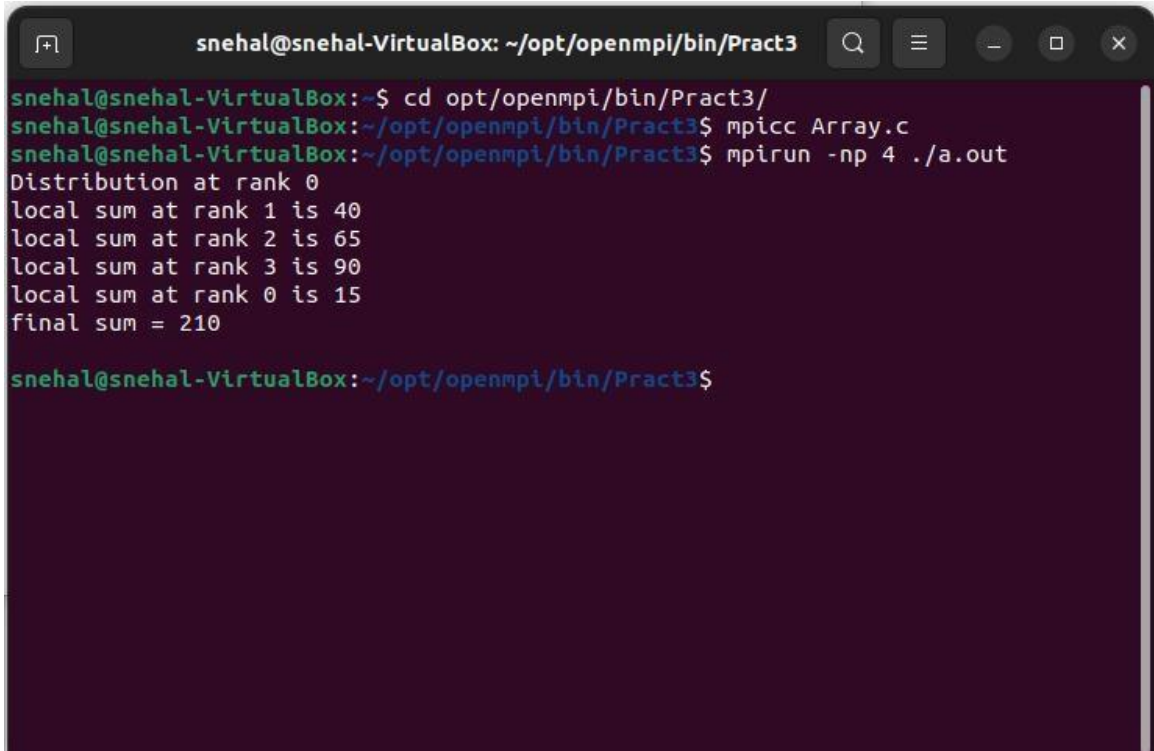
Code: Add 20 numbers in an array using 4 cores

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char* argv[])
{
    int rank, size;
    int num[20]; //N=20, n=4
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    for(int i=0;i<20;i++)
        num[i]=i+1;
    if(rank == 0){
        int s[4];
        printf("Distribution at rank %d \n", rank);
        for(int i=1;i<4;i++)
            MPI_Send(&num[i*5], 5, MPI_INT, i, 1, MPI_COMM_WORLD); //N/n i.e. 20/4=5
        int sum=0, local_sum=0;
        for(int i=0;i<5;i++)
        {
            local_sum=local_sum+num[i];
        }
        for(int i=1;i<4;i++)
        {
            MPI_Recv(&s[i], 1, MPI_INT, i, 1, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
        }
        printf("local sum at rank %d is %d\n", rank,local_sum);

        sum=local_sum;
```

```
for(int i=1;i<4;i++)
sum=sum+s[i];
printf("final sum = %d\n\n",sum);
}else
{
int k[5];
MPI_Recv(k, 5, MPI_INT, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
int local_sum=0;
for(int i=0;i<5;i++)
{
local_sum=local_sum+k[i];
}
printf("local sum at rank %d is %d\n", rank, local_sum);
MPI_Send(&local_sum, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
}
MPI_Finalize();
return 0;
}
```

Output:



```
snehal@snehal-VirtualBox: ~/opt/openmpi/bin/Pract3
snehal@snehal-VirtualBox:~$ cd opt/openmpi/bin/Pract3/
snehal@snehal-VirtualBox:~/opt/openmpi/bin/Pract3$ mpicc Array.c
snehal@snehal-VirtualBox:~/opt/openmpi/bin/Pract3$ mpirun -np 4 ./a.out
Distribution at rank 0
local sum at rank 1 is 40
local sum at rank 2 is 65
local sum at rank 3 is 90
local sum at rank 0 is 15
final sum = 210

snehal@snehal-VirtualBox:~/opt/openmpi/bin/Pract3$
```

RESULT:

Successfully implemented average number sum calculation.

ASSIGNMENT NO:4

AIM:

Implement Berkeley algorithm for clock synchronization.

Objective:

Ignoring significant outliers in calculation of average time difference

Outcome:

To improvise in accuracy of Cristian's algorithm

Explanation:

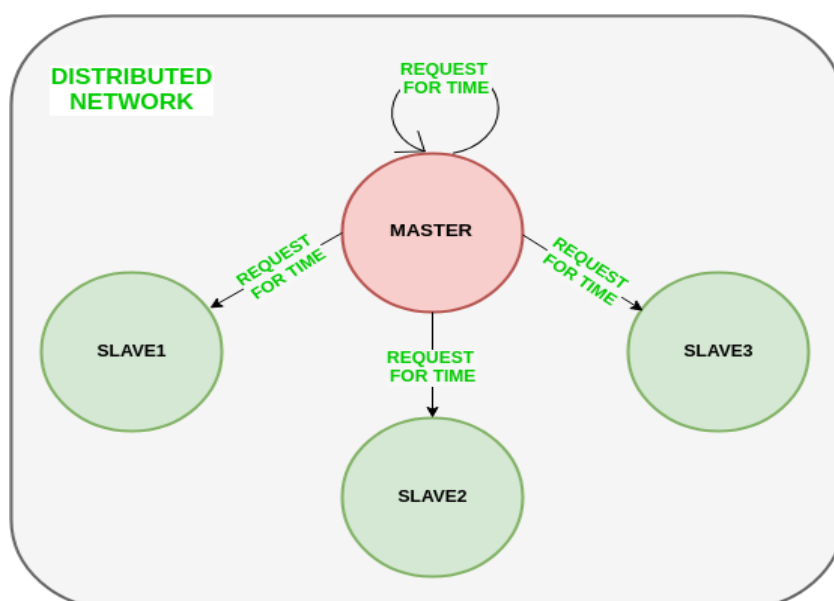
Berkeley's Algorithm is a clock synchronization technique used in distributed systems. The algorithm assumes that each machine node in the network either doesn't have an accurate time source or doesn't possess a UTC server.

Algorithm

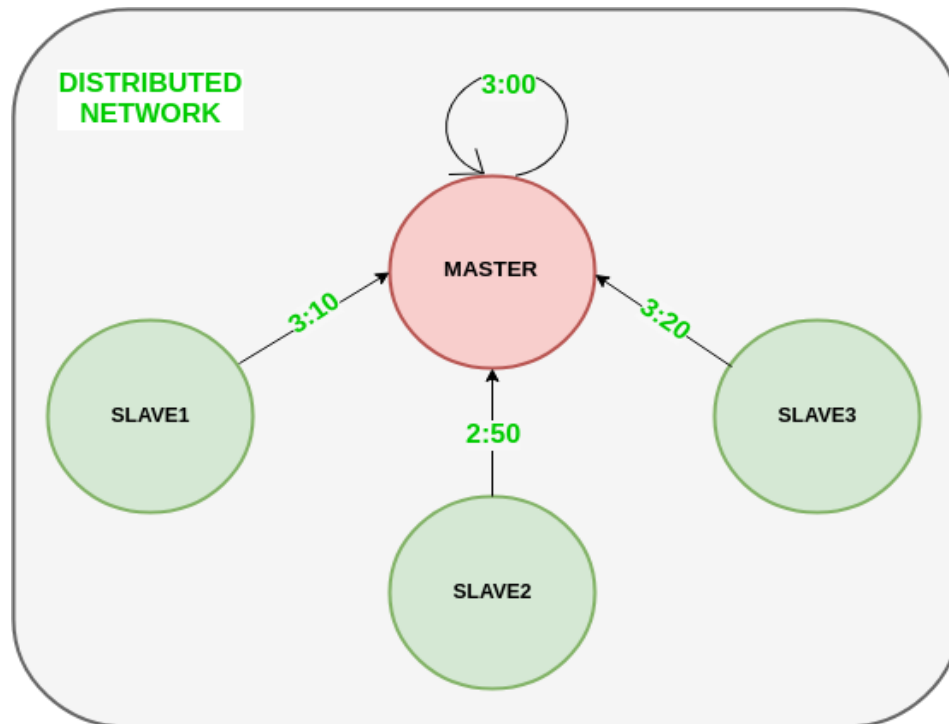
An individual node is chosen as the master node from a pool node in the network. This node is the main node in the network which acts as a master and the rest of the nodes act as slaves. The master node is chosen using an election process/leader election algorithm.

Master node periodically pings slave nodes and fetches clock time at them using Cristian's algorithm.

The diagram below illustrates how the master sends requests to slave nodes.



The diagram below illustrates how slave nodes send back time given by their system clock.



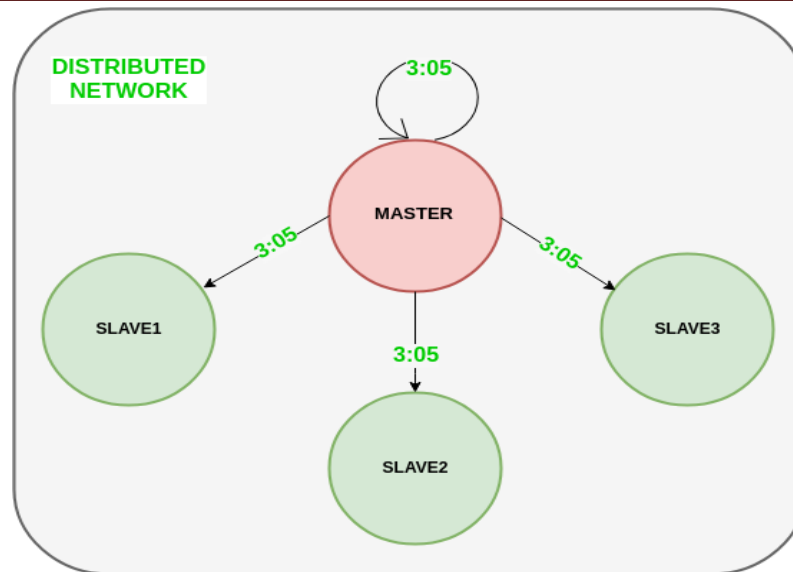
Master node calculates the average time difference between all the clock times received and the clock time given by the master's system clock itself. This average time difference is added to the current time at the master's system clock and broadcasted over the network.

The diagram below illustrates the last step of Berkeley's algorithm. Scope of Improvement
Improvision inaccuracy of Cristian's algorithm.

Ignoring significant outliers in the calculation of average time difference

In case the master node fails/corrupts, a secondary leader must be ready/pre-chosen to take the place of the master node to reduce downtime caused due to the master's unavailability.

Instead of sending the synchronized time, master broadcasts relative inverse time difference, which leads to a decrease in latency induced by traversal time in the network while the time of calculation at slave node.



Features of Berkeley's Algorithm:

Centralized time coordinator: Berkeley's Algorithm uses a centralized time coordinator, which is responsible for maintaining the global time and distributing it to all the client machines.

Clock adjustment: The algorithm adjusts the clock of each client machine based on the difference between its local time and the time received from the time coordinator.

Average calculation: The algorithm calculates the average time difference between the client machines and the time coordinator to reduce the effect of any clock drift.

Fault tolerance: Berkeley's Algorithm is fault-tolerant, as it can handle failures in the network or the time coordinator by using backup time coordinators.

Accuracy: The algorithm provides accurate time synchronization across all the client machines, reducing the chances of errors due to time discrepancies.

Scalability: The algorithm is scalable, as it can handle a large number of client machines, and the time coordinator can be easily replicated to provide high availability.

Security: Berkeley's Algorithm provides security mechanisms such as authentication and encryption to protect the time information from unauthorized access or tampering.

The code below is a c++ script that can be used to trigger a master clock server.

server.cpp:

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <arpa/inet.h>
#include <vector>
#include <cstdlib>
#include <ctime>

#define PORT 8080
using namespace std;

// function for string delimiter
vector<string> split(string s, string delimiter) {
    size_t pos_start = 0, pos_end, delim_len = delimiter.length();
    string token;
    vector<string> res;

    while ((pos_end = s.find (delimiter, pos_start)) != string::npos) {
        token = s.substr (pos_start, pos_end - pos_start);
        pos_start = pos_end + delim_len;
        res.push_back (token);
    }
    res.push_back (s.substr (pos_start));
    return res;
}

int main(int argc, char *argv[])
{
    // /* deal with input arguments*/
    // std::cout << "print arguments:\nargc == " << argc << '\n';
    // for(int ndx{}; ndx != argc; ++ndx) {
    //     std::cout << "argv[" << ndx << "] == " << argv[ndx] << '\n';
    // }
    // std::cout << "argv[" << argc << "] == "
```

```
//      << static_cast<void*>(argv[argc]) << '\n';
srand((unsigned int)time(NULL)); // avoid always same output of rand()
float server_local_clock = rand() % 10; // range from 0 to 9
vector<float> clients_local_clocks;
printf("Sever starts. Server pid is %d \n", getpid());
printf("Server local clock is %f \n\n", server_local_clock);

// Socket Cite: https://www.geeksforgeeks.org/socket-programming-cc/?ref=lbp
int server_socket_fd, new_socket, valread;
vector<int> client_sockets;
vector<string> client_ips;
vector<int> client_ports;
struct sockaddr_in server_address;
server_address.sin_family = AF_INET; // IPv4
server_address.sin_addr.s_addr = INADDR_ANY; // localhost
server_address.sin_port = htons( PORT ); // 8080
int opt = 1; // for setsockopt

// Creating socket file descriptor (IPv4, TCP, IP)
if ((server_socket_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
{
    perror("Server: socket failed");
    exit(EXIT_FAILURE);
}

// Optional: it helps in reuse of address and port. Prevents error such as: "address
already in use".
if (setsockopt(server_socket_fd, SOL_SOCKET, SO_REUSEADDR |
SO_REUSEPORT, &opt, sizeof(opt)))
{
    perror("Server: setsockopt");
    exit(EXIT_FAILURE);
}

// Forcefully attaching socket to the port 8080
if (bind(server_socket_fd, (struct sockaddr *)&server_address,
        sizeof(server_address))<0)
{
    perror("Server: bind failed");
    exit(EXIT_FAILURE);
}

// Putting the server socket in a passive mode, waiting for the client to approach the
server to make a connection
```

```
// The backlog=7, defines the maximum length to which the queue of pending
connections for sockfd may grow.
// If a connection request arrives when the queue is full, the client may receive an
error with an indication of ECONNREFUSED.
if (listen(server_socket_fd, 7) < 0)
{
    perror("Server: listen");
    exit(EXIT_FAILURE);
}
printf("Server: server is listening ...\n\nYou can open one or multiple new terminal
windows now to run ./client\n");
int clients_ctr = 0;

// Setting up buffer for receiving msg
char recv_buf[65536];
memset(recv_buf, '\0', sizeof(recv_buf));

int in_client_enough = 0;
while ( in_client_enough == 0) { // block on accept() until positive fd or error
    struct sockaddr_in client_addr;
    socklen_t length = sizeof(client_addr);
    // Extracting the first connection request on the queue of pending connections for
the listening socket (server_socket_fd)
    // Creates a new connected socket, and returns a new file descriptor referring to
that socket
    if ((new_socket = accept(server_socket_fd, (struct sockaddr *)&client_addr,
        (socklen_t *)&length)) < 0)
    {
        perror("Server: accept");
        exit(EXIT_FAILURE);
    }

    clients_ctr ++;
    printf("\nYou have connected %d client(s) now.", clients_ctr);

    // converting the network address structure src in the af address family into a
character string.
    char client_ip[INET_ADDRSTRLEN] = "";
    inet_ntop(AF_INET, &client_addr.sin_addr, client_ip, INET_ADDRSTRLEN);
    printf("Server: new client accepted. client ip and port: %s:%d\n", client_ip,
    ntohs(client_addr.sin_port));

    // store new client connection into array
```

```
    client_sockets.push_back(new_socket);
    client_ips.push_back(client_ip);
    client_ports.push_back(ntohs(client_addr.sin_port));

    printf("current connected clients amount is %d \n", int(client_sockets.size() ));

    cout << "Do you have enough clients? (please input '1' for yes, '0' for no):" ;
    cin >> in_client_enough;
    if (in_client_enough == 0){
        cout << "OK. Please continue opening one or multiple new terminal windows
to run ./client\n" << endl;
    }else if (in_client_enough != 1){
        cout << "Unrecognized input has been considered as 0. You can create one
more client.\n" << endl;
        in_client_enough = 0;
    }
}

printf("\nClients creation finished! There are totally %d connected clients.\n",
int(client_sockets.size() ));
printf("Asking all clients to report their local clock value ... \n\n\n");

for (int i = 0; i < client_sockets.size(); i++){
    // sending a message to client
    const char *msg = "Hello from server, please tell me your local clock value.";
    send(client_sockets[i], msg, strlen(msg), 0 );
    printf("Server: sent to client(%s:%d): %s\n", client_ips[i].c_str(), client_ports[i],
msg);

    // receiving
    while(recv(client_sockets[i], recv_buf, sizeof(recv_buf), 0) > 0 ){
        printf("Server:  recv  from  client(%s:%d):  '%s'  \n",  client_ips[i].c_str(),
client_ports[i], recv_buf);

        // convert char array to string
        string recv_msg = string(recv_buf);

        if (recv_msg.find("Hello  from  client,  my  local  clock  value  is") !=
string::npos){
            string substr_after_last_space;
            vector<string> split_str = split(recv_msg, " ");
            substr_after_last_space = split_str[ split_str.size() - 1 ];
```

```
        cout << "Server: received client local clock (string) is " <<
substr_after_last_space << endl;
        float substr_after_last_space_f = stof(substr_after_last_space);
        cout << "Server: received client local clock (float) is " <<
substr_after_last_space_f << endl;
        clients_local_clocks.push_back(substr_after_last_space_f);
    }
    memset(recv_buf, '\0', strlen(recv_buf));
    break;
}
}
printf("\n\n");

// average clock values
float all_clock_sum = server_local_clock;
for (int i = 0; i < clients_local_clocks.size(); i++){
    all_clock_sum += clients_local_clocks[i];
}
float avg_clock = all_clock_sum / (client_sockets.size() + 1);

// tell clients how to adjust
for (int i = 0; i < client_sockets.size(); i++){
    // prepare msg
    float offset = clients_local_clocks[i] - avg_clock;
    string operation;
    if (offset >= 0){
        operation = "minus";
    }else{
        operation = "add";
        offset = 0 - offset;
    }
    string msg_str = "From server, your clock adjustment offset is " + operation + " "
+ to_string(offset);
    char msg_char_array[msg_str.length() + 1];
    strcpy(msg_char_array, msg_str.c_str());
    // sending a message to client
    send(client_sockets[i], &msg_char_array, strlen(msg_char_array), 0);
    printf("Server: sent to client(%s:%d): %s\n", client_ips[i].c_str(), client_ports[i],
msg_char_array);
}

// adjust self
server_local_clock += avg_clock - server_local_clock;
```

```
printf("\n\nServer new local clock is %f\n\n", server_local_clock);
printf("Server: server stopped. \n");
close(server_socket_fd);
return 0;
}
```

client.cpp:

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <iostream>
#include <stdlib.h> /* srand, rand */
#include <cstdlib>
#include <ctime>
#include <vector>

#define PORT 8080
using namespace std;

// function for string delimiter
vector<string> split(string s, string delimiter) {
    size_t pos_start = 0, pos_end, delim_len = delimiter.length();
    string token;
    vector<string> res;

    while ((pos_end = s.find (delimiter, pos_start)) != string::npos) {
        token = s.substr (pos_start, pos_end - pos_start);
        pos_start = pos_end + delim_len;
        res.push_back (token);
    }

    res.push_back (s.substr (pos_start));
    return res;
}

int main(int argc, char const *argv[]){
    srand((unsigned int)time(NULL)); // avoid always same output of rand()
    float client_local_clock = rand() % 10; // range from 0 to 9
    printf("Client starts. Client pid is %d \n", getpid());
    printf("Client local clock is %f\n\n", client_local_clock);
```

```
int client_socket_fd, valread;
char client_read_buffer[1024] = {0};

struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
// server_addr.sin_addr.s_addr = inet_addr(argv[1]); // hardcode to 127.0.0.1
server_addr.sin_port = htons(PORT);

// Creating socket file descriptor (IPv4, TCP, IP)
if ((client_socket_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    printf("\n Client: Socket creation error \n");
    return -1;
}

// Converting IPv4 and IPv6 addresses from text to binary form,
// from character string src into a network
// address structure in the af address family, then copies the
// network address structure to dst.
if(inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr)<=0)
{
    printf("\nClient: Invalid address/ Address not supported \n");
    return -1;
}

// Connecting server, return 0 with success, return -1 with error
if (connect(client_socket_fd, (struct sockaddr *)&server_addr, sizeof(server_addr))
< 0)
{
    printf("\nClient: Connection Failed \n");
    return -1;
}

char server_ip[INET_ADDRSTRLEN]="";
inet_ntop(AF_INET, &server_addr.sin_addr, server_ip, INET_ADDRSTRLEN);
printf("Client:      connected      server(%s:%d).      \n",      server_ip,
ntohs(server_addr.sin_port));
printf("\n\n");

//
// first round communicattion
//

// receiving form server
```

Laboratory Practice -V (414454)**Class: BE(IT)**

```
valread = read( client_socket_fd , client_read_buffer, 1024);
```

```
printf("Client: read: '%s'\n",client_read_buffer );

// convert char array to string
string recv_msg = string(client_read_buffer);

// reply according to what client receive
if (strcmp(client_read_buffer, "Hello from server, please tell me your local clock
value.") == 0) {
    // prepare msg
    string msg_str = "Hello from client, my local clock value is " +
to_string(client_local_clock);
    char msg_char_array[msg_str.length() + 1];
    strcpy(msg_char_array, msg_str.c_str());
    // sending a message to server
    send(client_socket_fd , &msg_char_array , strlen(msg_char_array) , 0 );
    printf("Client: sent message: '%s'\n", msg_char_array);
}

//
// second round communicattion
//

// receiving form server
valread = read( client_socket_fd , client_read_buffer, 1024);
printf("Client: read: '%s'\n",client_read_buffer );

// convert char array to string
recv_msg = string(client_read_buffer);

if (recv_msg.find("From server, your clock adjustment offset is") != string::npos){
// if latter is a substring of former
    string substr_after_lastbutone_space;
    string substr_after_last_space;
    vector<string> split_str = split(recv_msg, " ");
    substr_after_lastbutone_space = split_str[ split_str.size() - 2 ];
    substr_after_last_space = split_str[ split_str.size() - 1 ];

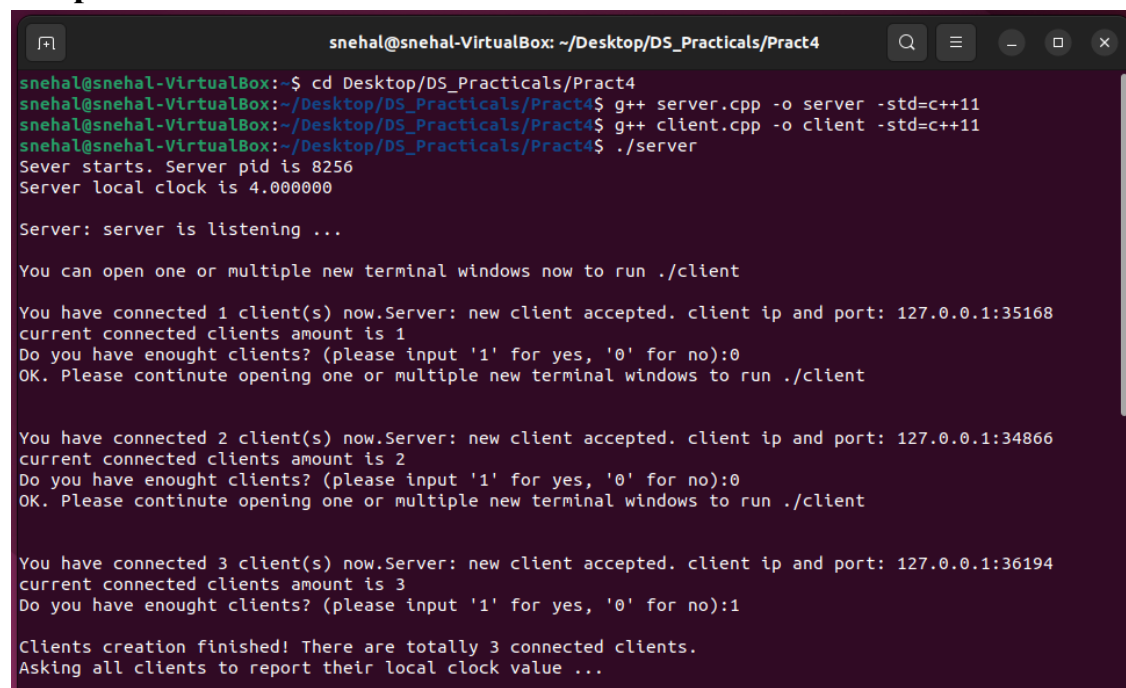
    cout << "Client: received local clock adjustment offset (string) is " <<
substr_after_lastbutone_space << " " << substr_after_last_space << endl;
    float substr_after_last_space_f = stof(substr_after_last_space);
    cout << "Client: received local clock adjustment offset (float) is " <<
substr_after_lastbutone_space << " " << substr_after_last_space_f << endl;
```

```
char oper_char_array[substr_after_lastbutone_space.length() + 1];
strcpy(oper_char_array, substr_after_lastbutone_space.c_str());
if (strcmp(oper_char_array, "add") == 0 ){
    client_local_clock += substr_after_last_space_f;
}else if (strcmp(oper_char_array, "minus") == 0 ){
    client_local_clock -= substr_after_last_space_f;
}

printf("Client local clock is %f\n\n", client_local_clock);
}

close(client_socket_fd);
return 0;
}
```

Output:



```
snehal@snehal-VirtualBox: ~/Desktop/DS_Practicals/Pract4
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract4$ g++ server.cpp -o server -std=c++11
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract4$ g++ client.cpp -o client -std=c++11
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract4$ ./server
Sever starts. Server pid is 8256
Server local clock is 4.000000

Server: server is listening ...

You can open one or multiple new terminal windows now to run ./client

You have connected 1 client(s) now.Server: new client accepted. client ip and port: 127.0.0.1:35168
current connected clients amount is 1
Do you have enough clients? (please input '1' for yes, '0' for no):0
OK. Please continue opening one or multiple new terminal windows to run ./client

You have connected 2 client(s) now.Server: new client accepted. client ip and port: 127.0.0.1:34866
current connected clients amount is 2
Do you have enough clients? (please input '1' for yes, '0' for no):0
OK. Please continue opening one or multiple new terminal windows to run ./client

You have connected 3 client(s) now.Server: new client accepted. client ip and port: 127.0.0.1:36194
current connected clients amount is 3
Do you have enough clients? (please input '1' for yes, '0' for no):1
Clients creation finished! There are totally 3 connected clients.
Asking all clients to report their local clock value ...
```

```
snehal@snehal-VirtualBox: ~/Desktop/DS_Practicals/Pract4
current connected clients amount is 3
Do you have enough clients? (please input '1' for yes, '0' for no):1

Clients creation finished! There are totally 3 connected clients.
Asking all clients to report their local clock value ...

Server: sent to client(127.0.0.1:35168): 'Hello from server, please tell me your local clock value.'
Server: rcv from client(127.0.0.1:35168): 'Hello from client, my local clock value is 9.000000'
Server: received client local clock (string) is 9.000000
Server: received client local clock (float) is 9
Server: sent to client(127.0.0.1:34866): 'Hello from server, please tell me your local clock value.'
Server: rcv from client(127.0.0.1:34866): 'Hello from client, my local clock value is 8.000000'
Server: received client local clock (string) is 8.000000
Server: received client local clock (float) is 8
Server: sent to client(127.0.0.1:36194): 'Hello from server, please tell me your local clock value.'
Server: rcv from client(127.0.0.1:36194): 'Hello from client, my local clock value is 0.000000'
Server: received client local clock (string) is 0.000000
Server: received client local clock (float) is 0

Server: sent to client(127.0.0.1:35168): 'From server, your clock adjustment offset is minus 3.750000'
Server: sent to client(127.0.0.1:34866): 'From server, your clock adjustment offset is minus 2.750000'
Server: sent to client(127.0.0.1:36194): 'From server, your clock adjustment offset is add 5.250000'

Server new local clock is 5.250000

Server: server stopped.
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract4$
```

```
snehal@snehal-VirtualBox: ~/Desktop/DS_Practicals/Pract4
snehal@snehal-VirtualBox:~$ cd Desktop/DS_Practicals/Pract4
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract4$ ./client
Client starts. Client pid is 8320
Client local clock is 8.000000

Client: connected server(127.0.0.1:8080).

Client: read: 'Hello from server, please tell me your local clock value.'
Client: sent message: 'Hello from client, my local clock value is 8.000000'
Client: read: 'From server, your clock adjustment offset is minus 2.750000'
Client: received local clock adjustment offset (string) is minus 2.750000
Client: received local clock adjustment offset (float) is minus 2.75
Client local clock is 5.250000

snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract4$
```

```
snehal@snehal-VirtualBox: ~/Desktop/DS_Practicals/Pract4
snehal@snehal-VirtualBox:~$ cd Desktop/DS_Practicals/Pract4
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract4$ ./client
Client starts. Client pid is 8290
Client local clock is 9.000000

Client: connected server(127.0.0.1:8080).

Client: read: 'Hello from server, please tell me your local clock value.'
Client: sent message: 'Hello from client, my local clock value is 9.000000'
Client: read: 'From server, your clock adjustment offset is minus 3.750000'
Client: received local clock adjustment offset (string) is minus 3.750000
Client: received local clock adjustment offset (float) is minus 3.75
Client local clock is 5.250000

snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract4$
```

```
snehal@snehal-VirtualBox: ~/Desktop/DS_Practicals/Pract4
snehal@snehal-VirtualBox:~$ cd Desktop/DS_Practicals/Pract4
snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract4$ ./client
Client starts. Client pid is 8346
Client local clock is 0.000000

Client: connected server(127.0.0.1:8080).

Client: read: 'Hello from server, please tell me your local clock value.'
Client: sent message: 'Hello from client, my local clock value is 0.000000'
Client: read: 'From server, your clock adjustment offset is add 5.250000'
Client: received local clock adjustment offset (string) is add 5.250000
Client: received local clock adjustment offset (float) is add 5.25
Client local clock is 5.250000

snehal@snehal-VirtualBox:~/Desktop/DS_Practicals/Pract4$
```

RESULT:

Maintaining the global time and distributing it to all the client machines.

ASSIGNMENT NO. – 5

AIM:

Implement token ring based mutual exclusion algorithm.

Objective:

To learn sequence number is used to distinguish old and current requests.

Outcome:

To the Site possesses the unique token, it is allowed to enter its critical section

Explanation:

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

Mutual exclusion in single computer system Vs. distributed system:

In single computer system, memory and other resources are shared between different processes. The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variable (For example: Semaphores) mutual exclusion problem can be easily solved.

In Distributed systems, we neither have shared memory nor a common physical clock and there for we can not solve mutual exclusion problem using shared variables. To eliminate the mutual exclusion problem in distributed system approach based on message passing is used.

A site in distributed system do not have complete information of state of the system due to lack of sharedmemory and a common physical clock.

Requirements of Mutual exclusion Algorithm:

No Deadlock:

Two or more site should not endlessly wait for any message that will never arrive.

No Starvation:

Every site who wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other siteare repeatedly executing critical section

Fairness:

Each site should get a fair chance to execute critical section. Any request to execute

critical section must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system.

Fault Tolerance:

In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

Solution to distributed mutual exclusion:

As we know shared variables or a local kernel can not be used to implement mutual exclusion in distributed systems. Message passing is a way to implement mutual exclusion. Below are the three approaches based on message passing to implement mutual exclusion in distributed systems:

1. Token Based Algorithm:

- A unique token is shared among all the sites.
- If a site possesses the unique token, it is allowed to enter its critical section
- This approach uses sequence number to order requests for the critical section.
- Each requests for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.
- This approach insures Mutual exclusion as the token is unique
- Example:
Suzuki-Kasami's Broadcast Algorithm

2. Non-token based approach:

- A site communicates with other sites in order to determine which sites should execute critical section next. This requires exchange of two or more successive round of messages among sites.
- This approach use timestamps instead of sequence number to order requests for the critical section.
- When ever a site make request for critical section, it gets a timestamp. Timestamp is also used to resolve any conflict between critical section requests.
- All algorithm which follows non-token based approach maintains a logical clock. Logical clocks get updated according to Lamport's scheme
- Example:
Lamport's algorithm, Ricart–Agrawala algorithm

3. Quorum based approach:

- Instead of requesting permission to execute the critical section from all other sites, Each site requests only a subset of sites which is called a quorum.
- Any two subsets of sites or Quorum contains a common site.
- This common site is responsible to ensure mutual exclusion
- Example:
Maekawa's Algorithm

1. MutualServer.java

```
import java.io.*;
import java.net.*;
public class MutualServer implements Runnable
{
    Socket socket=null; static
    ServerSocket ss;
    MutualServer(Socket newSocket)
    {
        this.socket=newSocket;
    }
    public static void main(String args[]) throws IOException
    {
        ss=new ServerSocket(7000);
        System.out.println("Server Started");
        while(true)
        {
            Socket s = ss.accept();
            MutualServer es = new MutualServer(s); Thread
            t = new Thread(es);
            t.start();
        }
    }
    public void run()
    {
        try
        {BufferedReader      in      =      new      BufferedReader(new
        InputStreamReader(socket.getInputStream()));
        while(true)
        {
            System.out.println(in.readLine());
        }
        }
        catch(Exception e){ }
    }
}
```

2. ClientOne.java

```
import java.io.*;
import java.net.*;
public class ClientOne
{
    public static void main(String args[])throws IOException
```



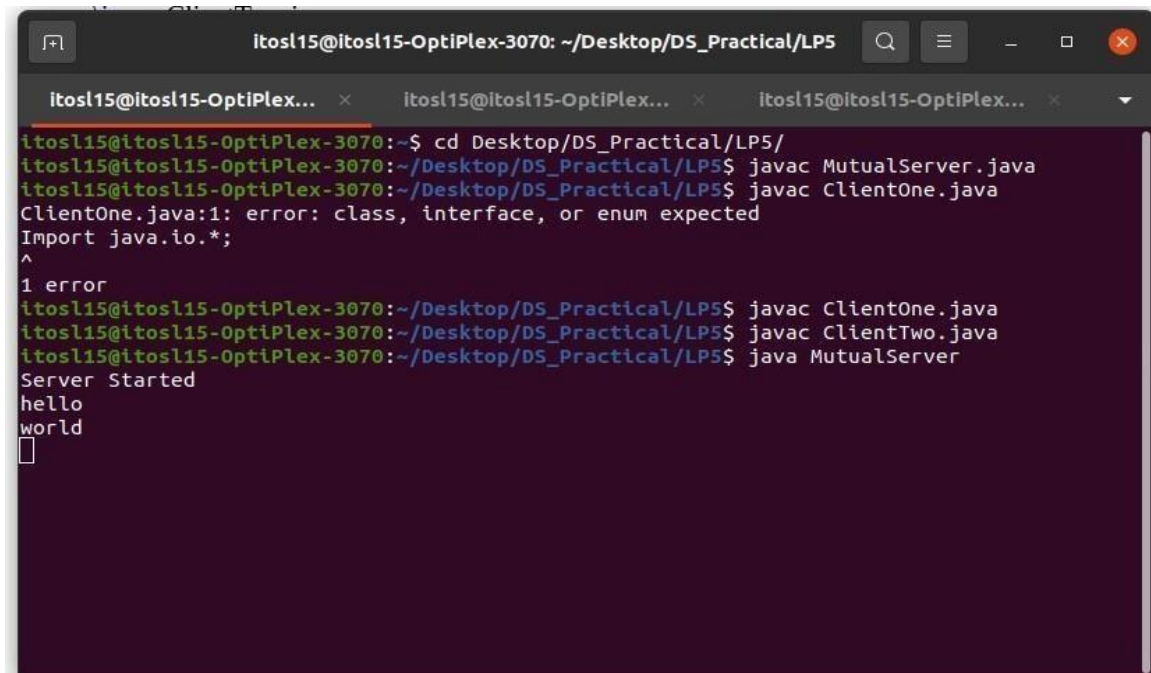
```
{
    Socket s=new Socket("localhost",7000);
    PrintStream out = new PrintStream(s.getOutputStream());
    ServerSocket ss = new ServerSocket(7001);
    Socket s1 = ss.accept();
    BufferedReader in1 = new BufferedReader(new
    InputStreamReader(s1.getInputStream()));
    PrintStream out1 = new PrintStream(s1.getOutputStream());
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String str="Token";
    while(true)
    {
        if(str.equalsIgnoreCase("Token"))
        {
            System.out.println("Do you want to send some data");
            System.out.println("Enter Yes or No"); str=br.readLine();
            if(str.equalsIgnoreCase("Yes"))
            {System.out.println("Enter the data");
            str=br.readLine();
            out.println(str);
            }
            out1.println("Token");
            }
            System.out.println("Waiting for Token");
            str=in1.readLine();
        }
    }
}
```

3. ClientTwo.java

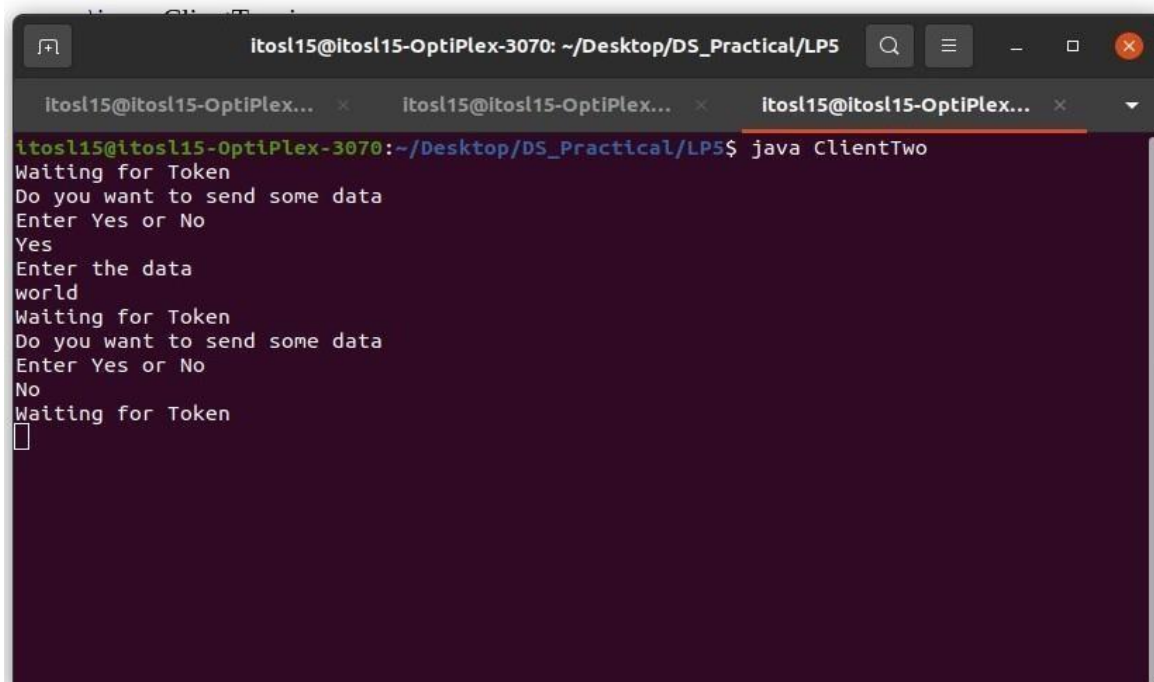
```
import java.io.*;
import java.net.*;
public class ClientTwo
{
    public static void main(String args[])throws IOException
    {
        Socket s=new Socket("localhost",7000);
        PrintStream out = new PrintStream(s.getOutputStream()); Socket
        s2=new Socket("localhost",7001); BufferedReader in2 = new
        BufferedReader(new InputStreamReader(s2.getInputStream()));
        PrintStream out2 = new PrintStream(s2.getOutputStream());
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String str;
```

```
while(true)
{
    System.out.println("Waiting for Token");
    str=in2.readLine();
    if(str.equalsIgnoreCase("Token"))
    {
        System.out.println("Do you want to send some data");
        System.out.println("Enter Yes or No"); str=br.readLine();
        if(str.equalsIgnoreCase("Yes")){
            System.out.println("Enter the data"); str=br.readLine();
            out.println(str);
        }
        out2.println("Token");
    }
}
}
```

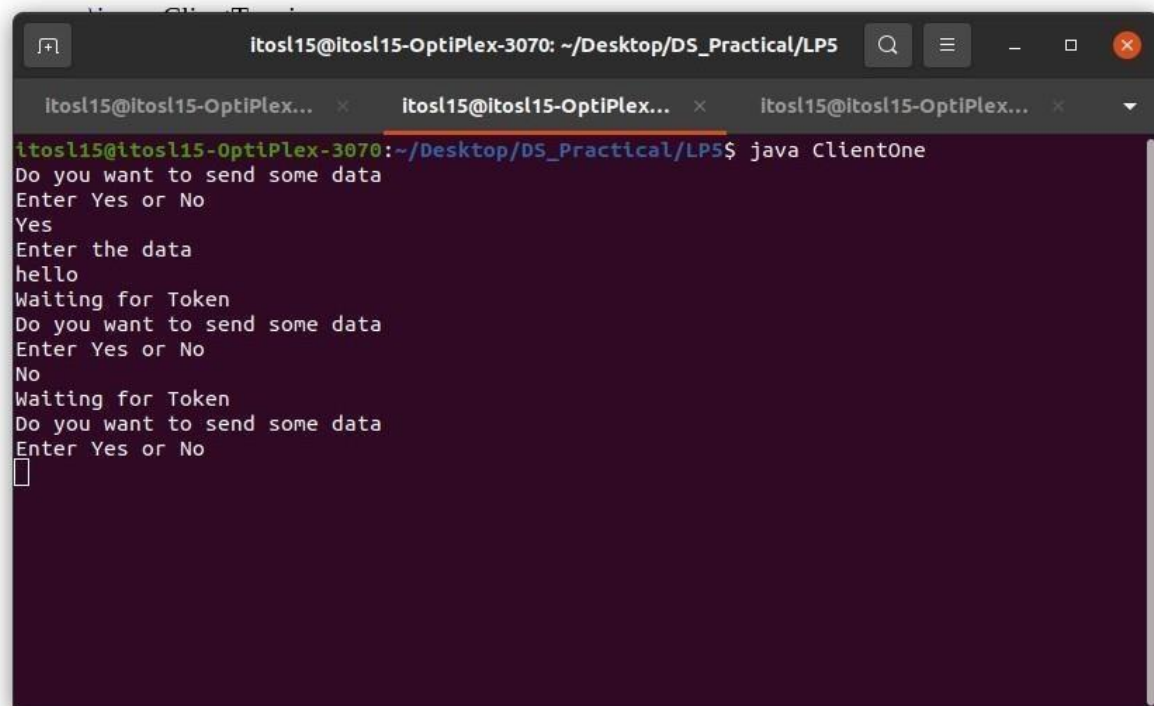
OUTPUT



```
itosl15@itosl15-OptiPlex-3070: ~/Desktop/DS_Practical/LP5
itosl15@itosl15-OptiPlex-3070:~$ cd Desktop/DS_Practical/LP5/
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/LP5$ javac MutualServer.java
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/LP5$ javac ClientOne.java
ClientOne.java:1: error: class, interface, or enum expected
Import java.io.*;
^
1 error
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/LP5$ javac ClientOne.java
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/LP5$ javac ClientTwo.java
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/LP5$ java MutualServer
Server Started
hello
world
□
```



```
itosl15@itosl15-OptiPlex-3070: ~/Desktop/DS_Practical/LP5
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/LP5$ java ClientTwo
Waiting for Token
Do you want to send some data
Enter Yes or No
Yes
Enter the data
world
Waiting for Token
Do you want to send some data
Enter Yes or No
No
Waiting for Token
█
```



```
itosl15@itosl15-OptiPlex-3070: ~/Desktop/DS_Practical/LP5
itosl15@itosl15-OptiPlex-3070:~/Desktop/DS_Practical/LP5$ java ClientOne
Do you want to send some data
Enter Yes or No
Yes
Enter the data
hello
Waiting for Token
Do you want to send some data
Enter Yes or No
No
Waiting for Token
Do you want to send some data
Enter Yes or No
█
```

Result:

Successfully communicate with multiple client machine from server.

ASSIGNMENT NO. 6

AIM:

Implement Bully and Ring algorithm for leader election.

Objective:

Coordinator that performs functions needed by other processes.

Outcome:

Current process P elects itself as a coordinator.

Explanation:

The **bully** algorithm is a type of **Election algorithm** which is mainly used for choosing a coordinate. In a distributed system, we need some election algorithms such as **bully** and **ring** to get a coordinator that performs functions needed by other processes.

Election algorithms select a single process from the processes that act as coordinator. A new process is selected when the selected coordinator process crashes due to some reasons. In order to determine the position where the new copy of coordinator should be restarted, the election algorithms are used.

It assumes that each process has a unique priority number in the system, so the highest priority process will be chosen first as a new coordinator. When the current use coordinator process crashes, it elects a new process having the highest priority number. We note that priority number and pass it to each active process in the distributed system.

The **Bully** election algorithm is as follows:

Let's assume that P is a process that sends a message to the coordinator.

It will assume that the coordinator process is failed when it doesn't receive any response from the coordinator within the time interval T.

An election message will be sent to all the active processes by process P along with the highest priority number.

If it will not receive any response within the time interval T, the current process P elects itself as a coordinator.

After selecting itself as a coordinator, it again sends a message that process P is elected as their new coordinator to all the processes having lower priority.

If process P will receive any response from another process Q within time T:

It again waits for time T to receive another response, i.e., it has been elected as coordinator from process Q.

If it doesn't receive any response within time T, it is assumed to have failed, and the algorithm is restarted.

Code:

BullyAlgo.java:

```
import java.io.*;
class BullyAlgo
{
    int cood,ch,crash; int
    prc[];
    public void election(int n) throws IOException
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("\nThe Coordinator Has Crashed!");
        int flag=1;
        while(flag==1)
        {
            crash=0;
            for(int i1=0;i1<n;i1++)
                if(prc[i1]==0) crash++;
            if(crash==n)
            {
                System.out.println("\n*** All Processes Are Crashed ***"); break;
            }
            else
            {
                System.out.println("\nEnter The Initiator"); int
                init=Integer.parseInt(br.readLine());
                if((init<1)||((init>n)||((prc[init-1]==0)))
                {
                    System.out.println("\nInvalid Initiator"); continue;
                }
                for(int i1=init-1;i1<n;i1++)
                    System.out.println("Process "+(i1+1)+" Called For Election"); System.out.println("");
            }
        }
    }
}
```

```
for(int i1=init-1;i1<n;i1++)
{
if(prc[i1]==0)
{
System.out.println("Process "+(i1+1)+ " Is Dead");
}
else
System.out.println("Process "+(i1+1)+" Is In");
}
for(int i1=n-1;i1>=0;i1--)
if(prc[i1]==1)
{
cood=(i1+1);
System.out.println("\n*** New Coordinator Is "+(cood)+" ***"); flag=0;
break;
}
}
}
}
public void Bully() throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter The Number Of Processes: ");
int n=Integer.parseInt(br.readLine());
prc=new int[n];
crash=0;
for(int i=0;i<n;i++)
prc[i]=1;
cood=n;
do
{
System.out.println("\n\t1. Crash A Process");
System.out.println("\t2. Recover A Process");
System.out.println("\t3. Display New Coordinator");
System.out.println("\t4. Exit");
ch=Integer.parseInt(br.readLine());
switch(ch)
{
case 1: System.out.println("\nEnter A Process To Crash"); int
cp=Integer.parseInt(br.readLine());
```

```
if((cp>n)|| (cp<1)){
    System.out.println("Invalid Process! Enter A Valid Process");
}
else if((prc[cp-1]==1)&&(cood!=cp))
{
    prc[cp-1]=0;
    System.out.println("\nProcess "+cp+ " Has Been Crashed");
}
else if((prc[cp-1]==1)&&(cood==cp))
{
    prc[cp-1]=0;
    election(n);
}
else
    System.out.println("\nProcess "+cp+" Is Already Crashed"); break;
case 2: System.out.println("\nCrashed Processes Are: \n"); for(int
i=0;i<n;i++)
{
    if(prc[i]==0)
        System.out.println(i+1);
    crash++;
}
System.out.println("Enter The Process You Want To Recover"); int
rp=Integer.parseInt(br.readLine());
if((rp<1)|| (rp>n))
    System.out.println("\nInvalid Process. Enter A Valid ID"); else
    if((prc[rp-1]==0)&&(rp>cood))
    {
        prc[rp-1]=1;
        System.out.println("\nProcess "+rp+" Has Recovered"); cood=rp;
        System.out.println("\nProcess "+rp+ " Is The New Coordinator");
    }
    else if(crash==n)
    {
        prc[rp-1]=1;
        cood=rp;
        System.out.println("\nProcess "+rp+ " Is The New Coordinator"); crash--;
    }
    else if((prc[rp-1]==0)&&(rp<cood))
    {
```



```
    prc[rp-1]=1;
    System.out.println("\nProcess "+rp+" Has Recovered");
}
else
    System.out.println("\nProcess "+rp+" Is Not A Crashed Process"); break;
case 3: System.out.println("\nCurrent Coordinator Is "+cood); break;
case 4: System.exit(0);
break;
default: System.out.println("\nInvalid Entry!"); break;
}
}
while(ch!=4);
}
public static void main(String args[]) throws IOException
{
    BullyAlgo ob=new BullyAlgo();
    ob.Bully();
}
}
```

Ring.java:

```
import java.util.Scanner;
public class Ring {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int temp, i, j;
        char str[] = new char[10];
        Rr proc[] = new Rr[10];
        // object initialisation
        for (i = 0; i < proc.length; i++)
            proc[i] = new Rr();
        // scanner used for getting input from console
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of process : ");
        int num = in.nextInt();
        // getting input from users
        for (i = 0; i < num; i++) {
            proc[i].index = i;
            System.out.println("Enter the id of process : ");
            proc[i].id = in.nextInt();
        }
    }
}
```

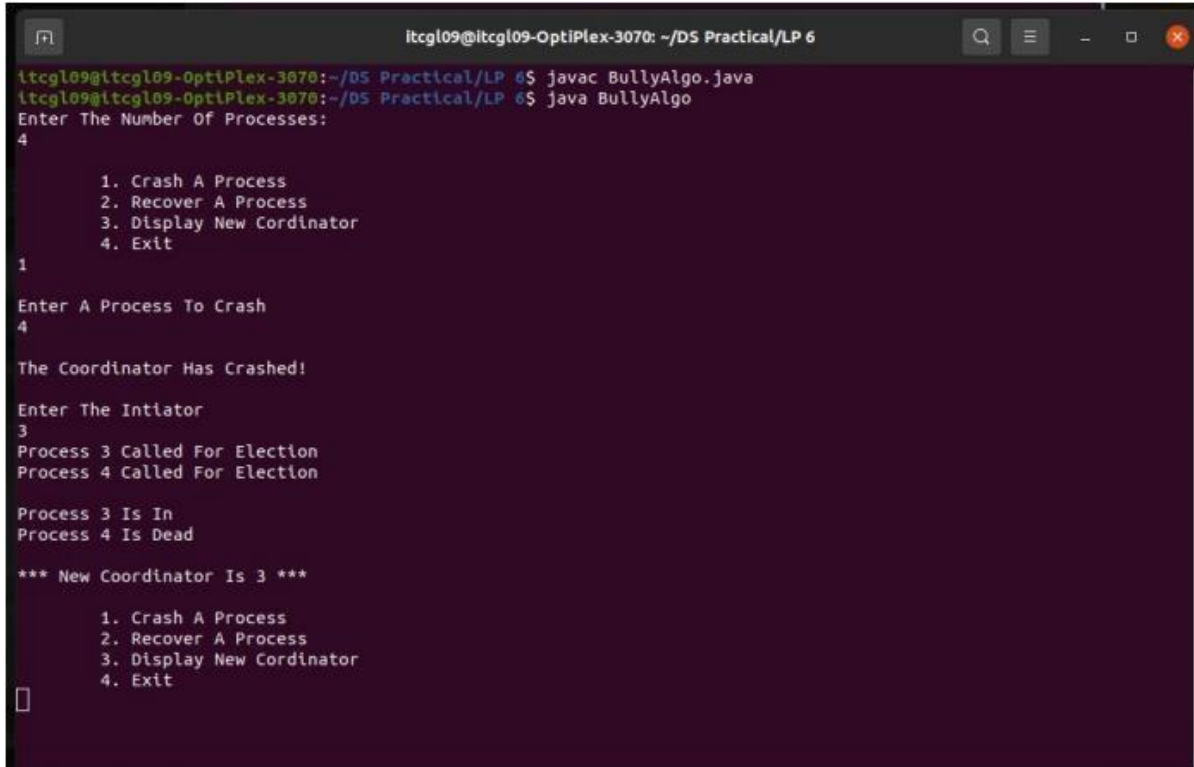
```
proc[i].state = "active";
proc[i].f = 0;
}
// sorting the processes from on the basis of id
for (i = 0; i < num - 1; i++) {
for (j = 0; j < num - 1; j++) {
if (proc[j].id > proc[j + 1].id) {
temp = proc[j].id;
proc[j].id = proc[j + 1].id;
proc[j + 1].id = temp;
}
}
}
for (i = 0; i < num; i++) {
System.out.print(" [" + i + "] " + "" + proc[i].id);
}
int init;
int ch;
int temp1;
int temp2;
int ch1;
int arr[] = new int[10];
proc[num - 1].state = "inactive";
System.out.println("\n process" + proc[num - 1].id + "select as co-ordinator");
while (true) {
System.out.println("\n 1.election 2.quit ");
ch = in.nextInt();
for (i = 0; i < num; i++) {
proc[i].f = 0;
}
switch (ch) {
case 1:
System.out.println("\n Enter the Process number who initialsied election : ");
init = in.nextInt();
temp2 = init;
temp1 = init + 1;
i = 0;
while (temp2 != temp1) {
if ("active".equals(proc[temp1].state) && proc[temp1].f == 0) {
System.out.println("\nProcess   " + proc[init].id + "send message to   " +
```

proc[temp1].id);

```
proc[temp1].f = 1;
init = temp1;
arr[i] = proc[temp1].id;
i++;
}
if (temp1 == num) {
temp1 = 0;
} else {
temp1++;
}
}
System.out.println("\nProcess" + proc[init].id + "sendmessage to " + proc[temp1].id);
arr[i] = proc[temp1].id;
i++;
int max = -1;
// finding maximum for co-ordinator selection
for (j = 0; j < i; j++) {
if (max < arr[j]) {
max = arr[j];
}
}
// co-ordinator is found then printing on console
System.out.println("\n process " + max + " select as co-ordinator");
for (i = 0; i < num; i++) {
if (proc[i].id == max) {
proc[i].state = "inactive";
}
}
break;
case 2:
System.out.println("Program terminated ...");
return ;
default:
System.out.println("\n invalid response \n");
break;
}
}
}
}
```

```
public int index; // to store the index of process
public int id; // to store id/name of process
public int f;
String state; // indicates whether active or inactive state of node
}
```

Output:



```
itcgl09@itcgl09-OptiPlex-3070: ~/DS Practical/LP 6
itcgl09@itcgl09-OptiPlex-3070:~/DS Practical/LP 6$ javac BullyAlgo.java
itcgl09@itcgl09-OptiPlex-3070:~/DS Practical/LP 6$ java BullyAlgo
Enter The Number Of Processes:
4
    1. Crash A Process
    2. Recover A Process
    3. Display New Coordinator
    4. Exit
1
Enter A Process To Crash
4
The Coordinator Has Crashed!
Enter The Initiator
3
Process 3 Called For Election
Process 4 Called For Election
Process 3 Is In
Process 4 Is Dead
*** New Coordinator Is 3 ***
    1. Crash A Process
    2. Recover A Process
    3. Display New Coordinator
    4. Exit
```

```
itcgl09@itcgl09-OptiPlex-3070: ~/DS Practical/LP 6
itcgl09@itcgl09-OptiPlex-3070:~/DS Practical/LP 6$ javac Ring.java
itcgl09@itcgl09-OptiPlex-3070:~/DS Practical/LP 6$ java Ring
Enter the number of process :
4
Enter the id of process :
1
Enter the id of process :
2
Enter the id of process :
3
Enter the id of process :
4
[0] 1 [1] 2 [2] 3 [3] 4
process4select as co-ordinator
1.election 2.quit
1
Enter the Process number who initialsted election :
3
Process 4send message to 1
Process 1send message to 2
Process 2send message to 3
Process3sendmessage to 4
process 4 select as co-ordinator
1.election 2.quit
2
Program terminated ...
itcgl09@itcgl09-OptiPlex-3070:~/DS Practical/LP 6$
```

Result:

You have executed all the processes

ASSIGNMENT NO:7

AIM:

Create a simple web service and write any distributed application to consume the web service.

Objective:

Web service can be created as a collection of open protocols and standards for exchanging information among systems or application

Outcome:

Web service can be published either on an intranet or the Internet

Explanation:

Service Provider or Publisher

This is the provider of the web service. The service provider implements the service and makes it available on the Internet or intranet.

We will write and publish a simple web service using .NET SDK.

Service Requestor or Consumer

This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

We will also write two web service requestors: one web-based consumer (ASP.NET application) and another Windows application-based consumer.

Given below is our first web service example which works as a service provider and exposes two methods (add and SayHello) as the web services to be used by applications. This is a standard template for a web service. .NET web services use the .asmx extension. Note that a method exposed as a web service has the WebMethod attribute. Save this file as FirstService.asmx in the IIS virtual directory (as explained in configuring IIS; for example, c:\MyWebSerces).

```
FirstService.aspx
<%@ WebService language = "C#" class = "FirstService" %>

using System;
using System.Web.Services;
using System.Xml.Serialization;

[WebService(Namespace = "http://localhost/MyWebServices/")]
```

```
public class FirstService : WebService{
    [WebMethod]
    public int Add(int a, int b) {
        return a + b;
    }

    [WebMethod]
    public String SayHello() {
        return "Hello World";
    }
}
```

To test a web service, it must be published. A web service can be published either on an intranet or the Internet. We will publish this web service on IIS running on a local machine. Let us start with configuring the IIS.

- Open Start → Settings → Control Panel → Administrative tools → Internet Services Manager.
- Expand and right-click on the default web site; select New → VirtualDirectory. The Virtual Directory Creation Wizard opens. Click Next.
- The "Virtual Directory Alias" screen opens. Type the virtual directoryname. For example, MyWebServices. Click Next.
- The "Web Site Content Directory" screen opens.
- Enter the directory path name for the virtual directory. For example, c:\MyWebServices. Click Next.
- The "Access Permission" screen opens. Change the settings as per your requirements. Let us keep the default settings for this exercise.
- Click the Next button. It completes the IIS configuration.
- Click Finish to complete the configuration.

To test whether the IIS has been configured properly, copy an HTML file (For example, x.html) in the virtual directory (C:\MyWebServices) created above. Now, open Internet Explorer and type **http://localhost/MyWebServices/x.html**. It should open the x.html file.

Note – If it does not work, try replacing the localhost with the IP address of your machine. If it still does not work, check whether IIS is running; you may need to reconfigure the IIS and the Virtual Directory.

To test this web service, copy FirstService.asmx in the IIS virtual directory created above (C:\MyWebServices). Open the web service in Internet Explorer (<http://localhost/MyWebServices/FirstService.asmx>). It should open your web service page. The page should have links to two methods exposed as web services by our application. Congratulations! You have written your first web service!

Testing the Web Service

As we have just seen, writing web services is easy in the .NET Framework. Writing web service consumers is also easy in the .NET framework; however, it is a bit more involved. As said earlier, we will write two types of service consumers, one web-based and another Windows application-based consumer. Let us write our first web service consumer.

Web-Based Service Consumer

Write a web-based consumer as given below. Call it WebApp.aspx. Note that it is an ASP.NET application. Save this in the virtual directory of the web service (c:\MyWebServices\WebApp.aspx).

This application has two text fields that are used to get numbers from the user to be added. It has one button, Execute, that when clicked gets the Add and SayHello web services.


```
WebApp.aspx
<%@ Page Language = "C#" %>
<script runat = "server">
    void runSvc_Click(Object sender, EventArgs e) {
        FirstService mySvc = new FirstService();
        Label1.Text = mySvc.SayHello();
        Label2.Text = mySvc.Add(Int32.Parse(txtNum1.Text),
Int32.Parse(txtNum2.Text)).ToString();
    }
</script>

<html>
    <head> </head>

    <body>
        <form runat = "server">
            <p>
                <em>First Number to Add </em>:
                <asp:TextBox id = "txtNum1" runat = "server" Width = "43px">4<
/asp:TextBox>
            </p>

            <p>
                <em>Second Number To Add </em>:
                <asp:TextBox id = "txtNum2" runat = "server" Width = "44px">5</asp:TextBox>
            </p>

            <p>
                <strong><u>Web Service Result -</u></strong>
            </p>

            <p>
```

```
<em>Hello world Service</em> :
<asp:Label id = "Label1" runat = "server" Font-Underline = "True">Label<
/asp:Label>
</p>

<p>
<em>Add Service</em> :
& <asp:Label id = "Label2" runat = "server" Font-Underline =
"True">Label</asp:Label>
</p>

<p align = "left">
<asp:Button id = "runService" onclick = "runService_Click" runat = "server" Text =
"Execute"></asp:Button>
</p>
</form>
</body>
</html>
```

After the consumer is created, we need to create a proxy for the web service to be consumed. This work is done automatically by Visual Studio .NET for us when referencing a web service that has been added. Here are the steps to be followed –

Create a proxy for the Web Service to be consumed. The proxy is created using the WSDL utility supplied with the .NET SDK. This utility extracts information from the Web Service and creates a proxy. The proxy is valid only for a particular Web Service. If you need to consume other Web Services, you need to create a proxy for this service as well. Visual Studio .NET creates a proxy automatically for you when the Web Service reference is added. Create a proxy for the Web Service using the WSDL utility supplied with the .NET SDK. It will create FirstService.cs file in the current directory. We need to compile it to create FirstService.dll (proxy) for the WebService.

```
c:> WSDL http://localhost/MyWebServices/FirstService.asmx?WSDL
```

```
c:> csc /t:library FirstService.cs
```

Put the compiled proxy in the bin directory of the virtual directory of the Web Service (c:\MyWebServices\bin). Internet Information Services (IIS) looks for the proxy in this directory.

Create the service consumer, in the same way we already did. Note that an object of the Web Service proxy is instantiated in the consumer. This proxy takes care of interacting with the service.

Type the URL of the consumer in IE to test it (for example, <http://localhost/MyWebServices/WebApp.aspx>).

Windows Application-Based Web Service Consumer

Writing a Windows application-based web service consumer is the same as writing any other

Windows application. You only need to create the proxy (which we have already done) and reference this proxy when compiling the application. Following is our Windows application that uses the web service. This application creates a web service object (of course, proxy) and calls the SayHello, and Add methods on it.

```
WinApp.cs

using System;
using System.IO;

namespace SvcConsumer {
    class SvcEater {
        public static void Main(String[] args) {
            FirstService mySvc = new FirstService();
            Console.WriteLine("Calling Hello World Service: " + mySvc.SayHello());
            Console.WriteLine("Calling Add(2, 3) Service: " + mySvc.Add(2, 3).ToString());
        }
    }
}
```

Compile it using `c:\>csc /r:FirstService.dll WinApp.cs`. It will create WinApp.exe. Run it to test the application and the web service.

Now, the question arises: How can you be sure that this application is actually calling the web service?

It is simple to test. Stop your web server so that the web service cannot be contacted. Now, run the WinApp application. It will fire a runtime exception. Now, start the web server again. It should work.

Result:

Successfully Created simple web service & Distributed application

Assignment No: 8

AIM:

Mini Project (In group): A Distributed Application for Interactive Multiplayer Games

Objective:

To create single deployable artifact to run game servers by containerizing the applications.

Outcome:

Each game is run as a separate process (dedicated server) to prevent impacting other game instances by resource usage

Explanation:

This is the main part of engineering, we're going to create a schematic model of our mental model for the whole system. to start we talk about the chosen stack to know the limitations and costs.

Deployment and orchestration

We have a single deployable artifact to run game servers by containerizing the applications. But deploying, maintaining, and scaling these artifacts and services is hard work to do. The Kubernetes helps handle deployments, orchestrate running containers, and manage nodes.

Load Balancer

The game manager is scalable horizontally, which means that the k8s run multiple instances of it concurrently. Therefore, we need a load balancer to distribute the requests to the game manager pods. The Nginx ingress is used as a load balancer to handle incoming requests and proxy them to the services.

Game Client

I chose Unity3D for the game client. The Unity Game Engine uses C# as the programming script.

Game Server

The game server is an authoritative server, which means that each client input is processed and validated on the server-side, then the game client checks the server validated data with its predictions. In addition, each game is run as a separate process

(dedicated server) to prevent impacting other game instances by resource usage or corruption, for example, if a game is crashed, other games remain safe to continue running. The game server is a stateful server because it uses memory to store game states like players' coordination's and inputs, so it's not replaceable or scalable horizontally. Our game is a fast-paced multiplayer game and should use UDP protocol to stream the game states and receive the players' inputs in the fastest way. Also, we need a TCP tunnel to transfer the game events which must be received with acknowledgment and in order. The game world simulations and calculations are not complicated, so I chose golang to develop the game server.

Game Manager Service

I merged some services with the game manager for the greater good (make deployment and maintenance easy), thus it must handle a lot of work. I chose golang to develop the game manager service. The game manager app is a stateless service and could be scaled horizontally.

APIs

An HTTP server APIs to handle the client requests in the game menu and store.

A WebSocket handler to keep a long-living connection with the client in the main menu to send events.

Matchmaking

The matchmaking service is responsible for putting the players in a queue and categorizing them depending on latency and rank. The Redis fits with these conditions. It supports the atomic lock concept which can be used for race condition challenges in multiple game manager instances.

Database

A database is needed to store some data like cars, items, users items, game logs, and so on. To achieve this, I used MySQL to store relational data (users' customized car parts), for the game data MongoDB fits here, but for simplicity, I moved forward with the MySQL.

Kubernetes API

The game manager uses K8S API to create a new pod and run the game server container as a dedicated server.

Game Session Management

The session manager creates, caches, manages, and destroys the game sessions.

Event Broker

The services talk to each other using the event broker, like when two players are matched or the game server got ready. I've looked for an opportunity to test the KubeMQ and this situation seemed good for me to use it. I used the KubeMQ community version for this. Also, Kafka was a good option, but on this scale, I prefer not to move forward with it for this project because of RAM consumption and node resource limits to decrease the server costs.

Result:

Distributed Application for Interactive Multiplayer Games