Marathwada Mitra Mandal's

COLLEGE OF ENGINEERING

Karvenagar, PUNE – 411 052

Accredited with "A++" Grade by NAAC // Accredited by NBA (Mechanical Engg. & Electrical Engg.)
Recipient of "Best College Award 2019" by SPPU // Recognized under section 2(f) and 12B of UGC Act 1956

DEPARTMENT OF INFORMATION TECHNOLOGY

<u>LABORATORY MANUAL</u>

BE (INFORMATION TECHNOLOGY)

(SEMESTER – II)

4144544 - LAB PRACTICE - V

2019 Course

Preface

A Distributed System is a collection of autonomous computer systems that are physically separated but are connected by a centralized computer network that is equipped with distributed system software. The autonomous computers will communicate among each system by sharing resources and files and performing the tasks assigned to them.

Distributed systems are an important development for IT and computer science as an increasing number of related jobs are so massive and complex that it would be impossible for a single computer to handle them alone. But distributed computing offers additional advantages over traditional computing environments. Distributed systems reduce the risks involved with having a single point of failure, bolstering reliability and fault tolerance. Modern distributed systems are generally designed to be scalable in near real-time; also, you can spin up additional computing resources on the fly, increasing performance and further reducing time to completion.

Distributed systems have evolved over time, but today's most common implementations are largely designed to operate via the internet and, more specifically, the cloud. A distributed system begins with a task, such as rendering a video to create a finished product ready for release.

The final outcome of each Distributed System study is information that will help to implement Distributed System. Keeping this in mind, course outcomes and lab assignments are prepared. It gives a complete understanding of the Distributed System. Proper mapping is also done with Program Outcomes and Program Specific Outcomes. This clearly states the goals for the students in terms of their skill development in the Distributed System.

'येथे बहुतांचे हित ।'

Marathwada Mitra Mandal's
COLLEGE OF ENGINEERING
Karvenagar, Pune

Vision

To aspire for the Welfare of Society
through excellence in Science and Technology.

Mission

Our Mission is to

- **M**ould young talent for higher endeavours.

- **M**eet the challenges of globalization.

- **C**ommit for social progress with values and ethics.

- **O**rient faculty and students for research and development.

- **E**mphasize excellence in all disciplines.

Department of Information Technology

Vision

➢ To emerge as a centre of excellence in Information Technology education for enrichment of society.

Mission

➢ To cater industry with engineers having theoretical & practical background and competent IT skills.
➢ To pursue advanced knowledge in the field of information technology.
➢ To inculcate budding IT engineers with professional and interpersonal skills.

Program Educational Objectives (PEOs)

The students of Information Technology Program after passing out will possess:

➢ Adequate knowledge and skills in Information Technology for implementation of complex problems with innovative approaches.
➢ Inclination and technical competency towards professional growth in the field of Information Technology.
➢ Ethics and value based interpersonal skills to facilitate lifelong learning and societal contributions.

Program Outcomes (POs)

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs)

Information Technology Graduates will be able to:

1. Demonstrate and apply data management, analytical and security skills to deliver software systems.
2. Design and develop interactive and intelligent systems using principles of human computer interaction and machine learning techniques.

DEPARTMENT OF INFORMATION TECHNOLOGY

Course Code: 4144544

Course Name: Lab Practice - V (2019 Course)

Teaching Scheme: **Credits:** 04   **Examination Scheme:**

Practical: 4 hrs/week   TW: 50 Marks

                                                  Practical: 25 Marks

A. Course Outcome

Course Outcome	Statement
	At the end of the course, a student will be able to (write/install/solve/apply)
414454.1	Demonstrate knowledge of the core concepts and techniques in distributed systems.
414454.2	Learn how to apply principles of state-of-the-Art Distributed systems in practical application.
414454.3	Design, build and test application programs on distributed systems

B. CO-PO mapping

Course Outcome	Program outcomes											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
414454.1	3	2	3	2	3	-	-	-	-	-	-	2
414454.2	3	2	3	2	3	-	-	-	-	-	-	2
414454.3	3	2	3	2	3	-	-	-	-	-	-	2
Average	3	2	3	2	3	0	0	0	0	0	0	2

C. CO-PSO mapping

Course Outcome	Program Specific Outcomes	
	1	**2**
414454.1	1	-
414454.2	1	-
414454.3	1	-
Average	1	-

DEPARTMENT OF INFORMATION TECHNOLOGY

Sr. No.	Title of Assignment	PO Mapping	PSO Mapping
1	Implement multi-threaded client/server Process communication using RMI.	PO1, PO2, PO3, PO4, PO5, PO12	PSO1
2	Develop any distributed application using CORBA to demonstrate object brokering. (Calculator or String operations).	PO1, PO2, PO3, PO4, PO5, PO12	PSO1
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.	PO1, PO2, PO3, PO4, PO5, PO12	PSO1
4	Implement Berkeley algorithm for clock synchronization.	PO1, PO2, PO3, PO4, PO5, PO12	PSO1
5	Implement token ring based mutual exclusion algorithm.	PO1, PO2, PO3, PO4, PO5, PO12	PSO1
6	Implement Bully and Ring algorithm for leader election.	PO1, PO2, PO3, PO4, PO5, PO12	PSO1
7	Create a simple web service and write any distributed application to consume the web service.	PO1, PO2, PO3, PO4, PO5, PO12	PSO1
8	Mini Project (In group): A Distributed Application for Interactive Multiplayer Games	PO1, PO2, PO3, PO4, PO5, PO12	PSO1
9	To develop any distributed application using Messaging System in Publish-Subscribe	PO1, PO2, PO3, PO4, PO5, PO12	PSO1

Experiment No. 1

TITLE: - Multi-threaded client/server Process communication using RMI.

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

THEORY: -

Multithreading in the Server

RMI automatically will execute each client in a separate thread. There will only be one instance of your server object no matter how many clients, so you need to make sure that shared data is synchronized appropriately. In particular, any data that may be accessed by more than one client at a time must be synchronized.

Also note that you cannot depend on data in the server to remain the same between two successive calls from a client. In our rental car example from the first class, a client may get a list of cars in one call, and in a later call try to reserve the car. In the meantime, another client may have already reserved that car. The server must double-check all data coming from the client to protect against this sort of error. Also for this reason, the server should never pass to the client any direct pointers to its internal data structures (such as an array index), as that type of data is highly likely to change before the client tries to use it.

Multithreading can also introduce very difficult to find bugs into your program. The types of bugs introduced because of multithreading are called "race conditions". A race condition is a bug that is timing sensitive. In other words, the bug only happens when several conditions happen at exactly the same time. With multithreading, the possibility of race conditions increases.

Unfortunately, it's almost impossible to know if a multithreading program has bugs or not. Each thread runs at the same time as the other threads. A computer with a single CPU, however, cannot really run multiple instructions at the same time. So it runs instructions from one thread for a while, and then runs some instructions from another thread. You have no way of knowing exactly when a thread may be interrupted and another thread run.

Since RMI automatically runs client calls in separate threads, you do not need to use Thread or Runnable yourself.

Thread States

You still need to worry about thread states, however, in RMI. A thread can be in one of several states. These include:

Running -- The thread is currently executing.

Dead -- The thread has completely finished executing, and will never execute again. This typically means that the client call to the server has completed.

Ready -- The thread can execute, but is currently not executing. The CPU has switched to another thread and will get back to this one in time.

Blocked -- A blocked thread is waiting for some external event to finish. Examples would include a thread that is trying to read from a file. The thread will block at the call to read from the file, and not continue executing until the read is finished. In this way, a thread that starts an external task will allow other threads to run until that external task is finished.

Waiting -- A thread can wait for other threads to finish work it needs to use, by calling the wait () method. A thread that is waiting for other threads to finish some work will not execute until those threads call the notify () method.

Sleeping -- A thread can voluntarily put itself to sleep for a certain period of time. The thread will start executing again only after the given period of time has passed. The thread puts itself to sleep using the Thread.sleep () method. A thread should put itself to sleep if it only wants to execute every so often.

Controlling Threads

There are various methods for controlling threads. We've seen Thread.start () to begin a thread. Thread.sleep () allows a thread to sleep for a specific period of time.

There's also Thread.yield (), which allows a thread to voluntarily give up control of the CPU to allow other threads to execute. If there are no other threads to execute, the thread will continue immediately.

Java keeps a queue of threads called the "ready queue". These are threads that are in the Ready state. Threads go from the Running state into one of the other states via method calls. For example, a thread that is Running will go into the sleeping State when Thread.sleep () is called. When the time period for the sleep is over, the thread goes into the Ready state and is placed into the ready queue. Whenever Java needs to figure out which thread to execute next, it pulls a thread from the ready queue.

Here's a diagram of how threads move from state to state:
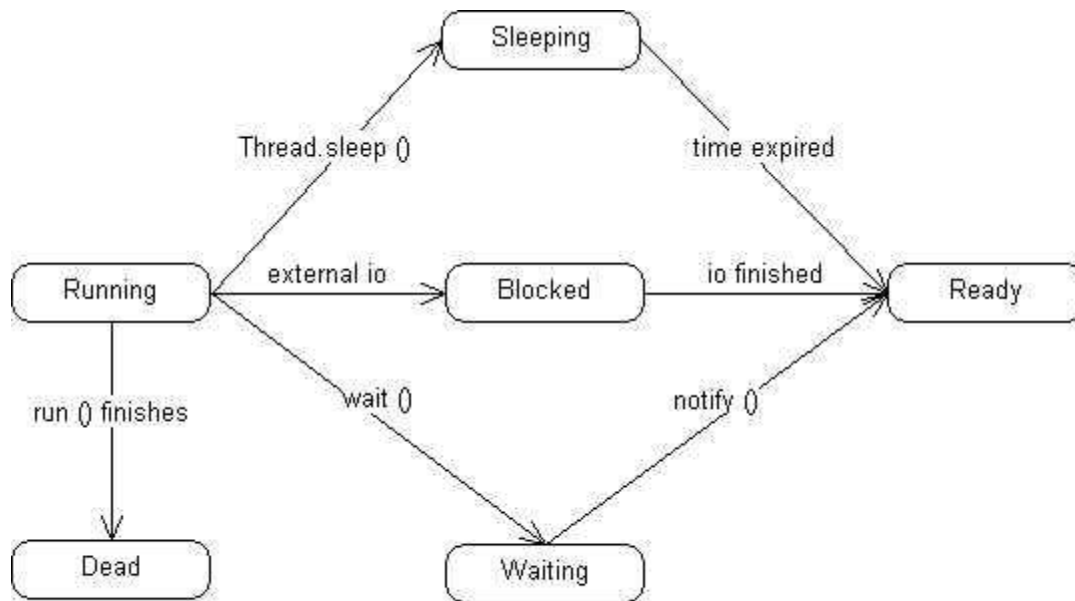
The thread transitions from Ready to Running when Java decides to run the thread some more. There's also a transition from Running directly to Ready when the thread uses the Thread.yield method to voluntarily give up control of the CPU.

There's a method called Thread.interrupt that can be used to interrupt a thread that is waiting, sleeping, or blocked. The thread will begin executing and will receive an InterruptedException.

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

  It initiates a connection with remote Virtual Machine (JVM),

  It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),

  It waits for the result

  It reads (unmarshals) the return value or exception, and

 It finally returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

It reads the parameter for the remote method

It invokes the method on the actual remote object, and

It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, a stub protocol was introduced that eliminates the need for skeletons.
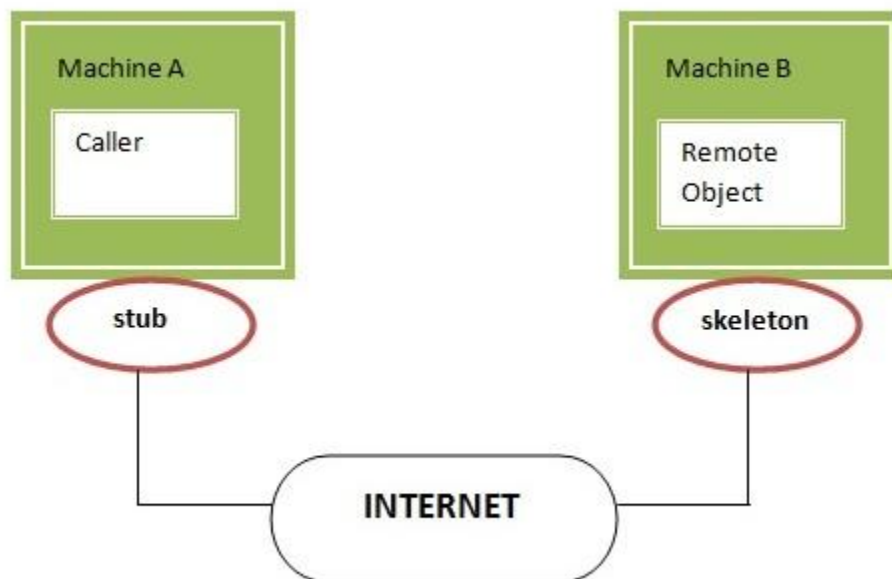
Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

.

The application need to locate the remote method

It need to provide the communication with the remote objects, and

The application needs to load the class definitions for the objects.

The RMI application has all these features, so it is called the distributed application.

Java RMI Example

This is given the 6 steps to write the RMI program.

- Create the remote interface
- Provide the implementation of the remote interface
- Compile the implementation class and create the stub and skeleton objects using the rmic tool
- Start the registry service by rmiregistry tool
- Create and start the remote application
- Create and start the client application

RMI Example

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.
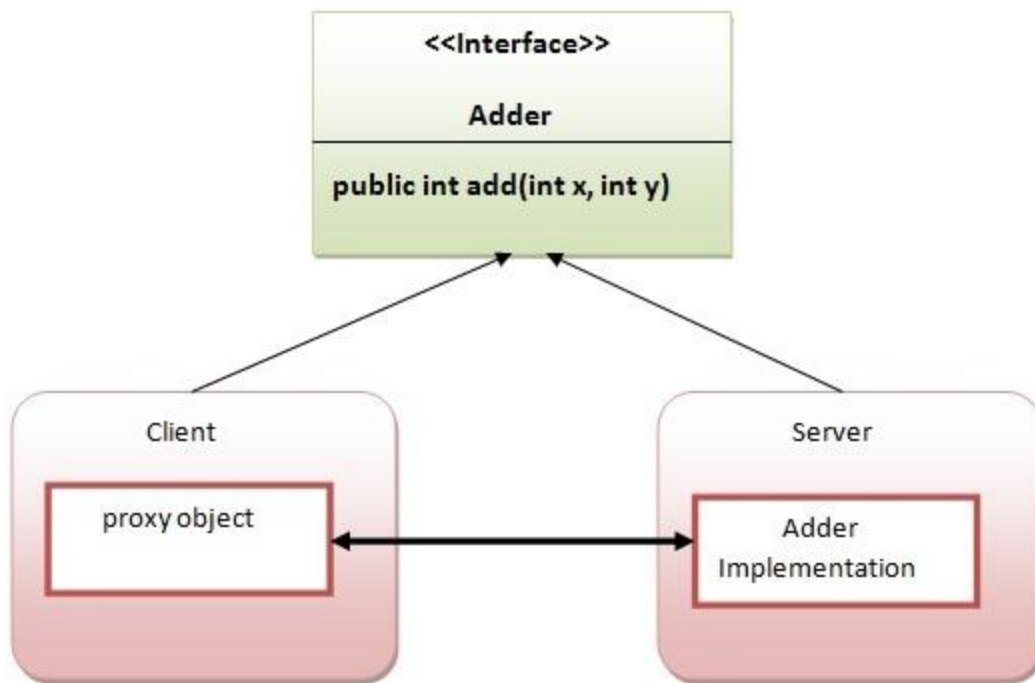
CONCLUSION:-

We have implemented Multi-threaded client/server Process communication using RMI

Questions:
1) Explain Process communication?
2) Define RMI ?
3) List basic principles of RMI architecture ?
4) Summarize layers of RMI Architecture ?
5) Classify the role of Remote Interface in RMI ?

Experiment No. 2

TITLE: - Distributed application using CORBA

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

THEORY: -

CORBA

The Common Object Request Broker Architecture (or CORBA) is an industry standard developed by the Object Management Group (OMG) to aid in distributed objects programming. It is important to note that CORBA is simply a specification. A CORBA implementation is known as an ORB (or Object Request Broker). There are several CORBA implementations available on the market such as VisiBroker, ORBIX, and others. JavaIDL is another implementation that comes as a core package with the JDK1.3 or above.

CORBA was designed to be platform and language independent. Therefore, CORBA objects can run on any platform, located anywhere on the network, and can be written in any language that has Interface Definition Language (IDL) mappings.

Similar to RMI, CORBA objects are specified with interfaces. Interfaces in CORBA, however, are specified in IDL. While IDL is similar to C++, it is important to note that IDL is not a programming language. For a detailed introduction to CORBA.

The Genesis of a CORBA Application

There are a number of steps involved in developing CORBA applications. These are:

Define an interface in IDL
Map the IDL interface to Java (done automatically)
Implement the interface
Develop the server
Develop a client
Run the naming service, the server, and the client.

We now explain each step by walking you through the development of a CORBA-based file transfer application, which is similar to the RMI application we developed earlier in this article. Here we will be using the JavaIDL, which is a core package of JDK1.3+.

Define the Interface

When defining a CORBA interface, think about the type of operations that the server will support. In the file transfer application, the client will invoke a method to download a file. Code Sample 5 shows the interface for FileInterface. Data is a new type introduced using the typedef keyword. A sequence in IDL is similar to an array except that a sequence does not have a fixed size. An octet is an 8-bit quantity that is equivalent to the Java type byte.

Note that the downloadFile method takes one parameter of type string that is declared in. IDL defines three parameter-passing modes: in (for input from client to server), out (for output from server to client), and inout (used for both input and output).

CORBA Features

CORBA achieves its flexibility in several ways:

It specifies an interface description language (IDL), that allows you to specify the interfaces to objects. IDL object interfaces describe, among other things:

The data that the object makes public.

The operations that the object can respond to, including the complete signature of the operation. CORBA operations are mapped to Java methods, and the IDL operation parameter types map to Java datatypes.

Exceptions that the object can throw. IDL exceptions are also mapped to Java exceptions, and the mapping is very direct.

CORBA provides bindings for many languages, including both non-object languages such as COBOL and C and object-oriented languages such as Smalltalk and Java.

All CORBA implementations provide an object request broker (ORB), that handles the routing of object requests in a way that is largely transparent to the application developer. For example, requests (method invocations) on remote objects that appear in the client code look just like local method invocations. The remote call functionality, including marshalling of parameter and return data, is taken care of for the programmer by the ORB.

CORBA specifies a network protocol, the Internet Inter-ORB Protocol (IIOP), that provides for transmission of ORB requests and data over a widely-available transport protocol: TCP/IP, the Internet standard.

There is a set of fully-specified services that ease the burden of application development by making it unnecessary for the developer to constantly reinvent the wheel. Among these services are:

> Naming. One or more services that let you resolve names that are bound to CORBA server objects.
>
> Transactions. Services that let you manage transaction control of data resources in a flexible and portable way.
>
> o   Events.

CORBA specifies over 12 services. Most of these are not yet implemented by CORBA ORB vendors.

The remainder of this section introduces some of the essential building blocks of an Oracle8i JServer CORBA application. These include:

the ORB--how to talk to remote objects
IDL--how to write a portable interface
the naming service (and JNDI)--how to locate a persistent object
object adapters--how to register a transient object
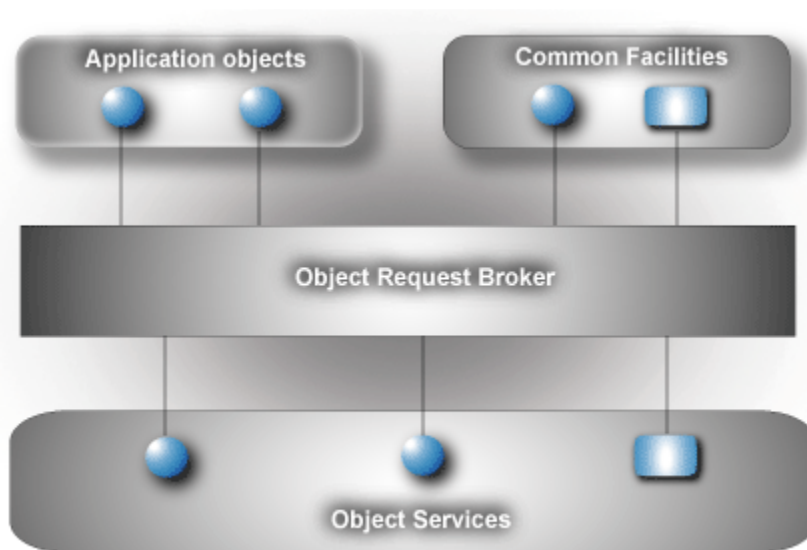
The basic steps for CORBA development include:

❶ Create the IDL to Define the Application Interfaces

The IDL provides the operating system and programming language independent interfaces to all services and components that are linked to the ORB. The IDL specifies a description of any services a server component exposes to the client. The term "IDL Compiler" is often used, but the IDL is actually translated into a programming language.

❷ Translate the IDL

An IDL translator typically generates two cooperative parts for the client and server implementation, stub code and skeleton code. The stub code generated for the interface classes is associated with a client application and provides the user with a well-defined Application Programming Interface (API). In this example, the IDL is translated into C++.

❸ Compile the Interface Files

Once the IDL is translated into the appropriate language, C++ in this example, these interface files are compiled and prepared for the object implementation.

❹ Complete the Implementation

If the implementation classes are incomplete, the spec and header files and complete bodies and definitions need to be modified before passing through to be compiled. The output is a complete client/server implementation.

❺ Compile the Implementation

Once the implementation class is complete, the client interfaces are ready to be used in the client application and can be immediately incorporated into the client process. This client process is responsible for obtaining an object reference to a specific object, allowing the client to make requests to that object in the form of a method call on its generated API.

❻ Link the Application

Once all the object code from steps three and five have been compiled, the object implementation classes need to be linked to the C++ linker. Once linked to the ORB library, in this example, ORBexpress, two executable operations are created, one for the client and one for the server.

❼ Run the Client and Server

The development process is now complete and the client will now communicate with the server. The server uses the object implementation classes allowing it to communicate with the objects created by the client requests.

CONCLUSION:-

We have developed a distributed application using CORBA to demonstrate object brokering. (Calculator or String operations).

Questions:
1) Explain Process communication?

2) Explain advantages of CORBA?

3) Describe CORBA?

4) List applications of Distributed systems.

Experiment No. 3

TITLE: - A distributed system, to find the sum of N elements in an array.

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.

THEORY: -

Message Passing Interface(MPI) is a 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 exchange information among these processes.

MPI uses two basic communication routines:

> **MPI_Send**, to send a message to another process.

> **MPI_Recv**, to receive a message from another process.

The syntax of MPI_Send and MPI_Recv is:

int MPI_Send(void *data_to_send,

> int send_count,

> MPI_Datatype send_type,

> int destination_ID,

> int tag,

> MPI_Comm comm);

int MPI_Recv(void *received_data,

> int receive_count,

> MPI_Datatype receive_type,

> int sender_ID,

int tag,

MPI_Comm comm,

MPI_Status *status);

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 the total sum of the array.

Given an array of integers, find sum of its elements.

Examples :

Input : arr[] = {1, 2, 3}

Output : 6

Explanation: 1 + 2 + 3 = 6

Input : arr[] = {15, 12, 13, 10}

Output : 50

MPI stands for Message Passing Interface. It is used to make communication between different processes in the same machine or across different machines in a network in a distributed environment. So that the task can be parallelized and work can be done faster. The MPI has made the parallelizing tasks very easy. It is available in the form of a standard library. The parallelism between different processes can be achieved with help of a rank.

The MPI library assigns the different processes with a unique number called a rank. The rank starts from 0 to n-1. If there are 4 processes then each process will be assigned a unique rank ranging from 0,1,2, and 3. These ranks can be used to differentiate and communicate between the different processes. You can read more about the MPI

programming at MPI Tutorials. In this post, we will see how we can compute the array sum using MPI programming.

MPI Programming

In order to start programming using MPI, you need to install the library first and then you can import the library and use it.

Installing MPI Library

You can download the MPI binary from the MPI Downloads. It is available for Linux Distro, Windows OS, and Mac OS. Then follow the instruction accordingly to install it on your machine. You can directly install it in Ubuntu. Just type *"mpicc"* in the terminal. It will give the command to install it. Copy the command and run it in the terminal.

You can include the mpi library in you C/C++ program by **"#include<mpi.h>"**.

A Simple MPI Program Structure

A simple MPI program written in C is given below to understand the MPI. The first process will print the "How are you?" message and the rest will print out " I am Fine".

#include<stdio.h>

#include<mpi.h>

int main(int argc, char *argv[]){

   int size, rank;

   MPI_Init(&argc, &argv);

   MPI_Comm_size(MPI_COMM_WORLD, &size);

   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   if(rank == 0){

```
    printf("Rank %d: How are you?\n", rank);

  }else{

    printf("Rank %d: I am Fine\n", rank);

  }

  MPI_Finalize();

  return 0;

}
```

CCopy

Some of the MPI library functions and variables that you can notice from the above code are as follows:

> MPI_Init() – It initialize the MPI
> MPI_Comm_size() – It determines the total number of processes in a group.
> MPI_Comm_rank() – It determines the id of the calling process.
> MPI_Finalize() – It binds the MPI program.
> MPI_COMM_WORLD – It is a communicator between processes.

How to Compile and Run MPI Program?

In order to compile the program, you need to run the *"mpicc"* command.

mpicc -o objectname programname.c

BashCopy

For example, if the program name is **"hello_mpi.c"**, then you can compile it as **"mpicc -o hello_mpi hello_mpi.c"**.

Now you can run the program with *"mpirun"* command. You can run literally on any number of processes.

mpirun -np total_process_number ./objectname

BashCopy

For example, you can run *"hello_mpi.*c" as *"mpirun -np 4 ./hello_mpi"* once the program code is successfully compiled.

CONCLUSION:-

We have developed 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.

Questions:
1) List advantages of message passing interface.

2) Explain Benefits of the message passing interface

3) Describe message passing interface (MPI)?

4) List applications of MPI.

Experiment No. 4

TITLE: - Berkeley algorithm for clock synchronization

AIM: - Implement Berkeley algorithm for clock synchronization

THEORY: -

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

1) 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.

2) Master node periodically pings slaves nodes and fetches clock time at them using Cristian's algorithm.
The diagram below illustrates how the master sends requests to slave nodes.

Berkeley's Algorithm is an algorithm that is used for clock Synchronization in distributed systems. This algorithm is used in cases when some or all systems of the distributed network have one of these issues –

      A. The machine does not have an accurate time source.
      B. The network or machine does not have a UTC server.

Distributed system contains multiple nodes that are physically separated but are linked together using a network.

DISTRIBUTED OPERATING SYSTEM
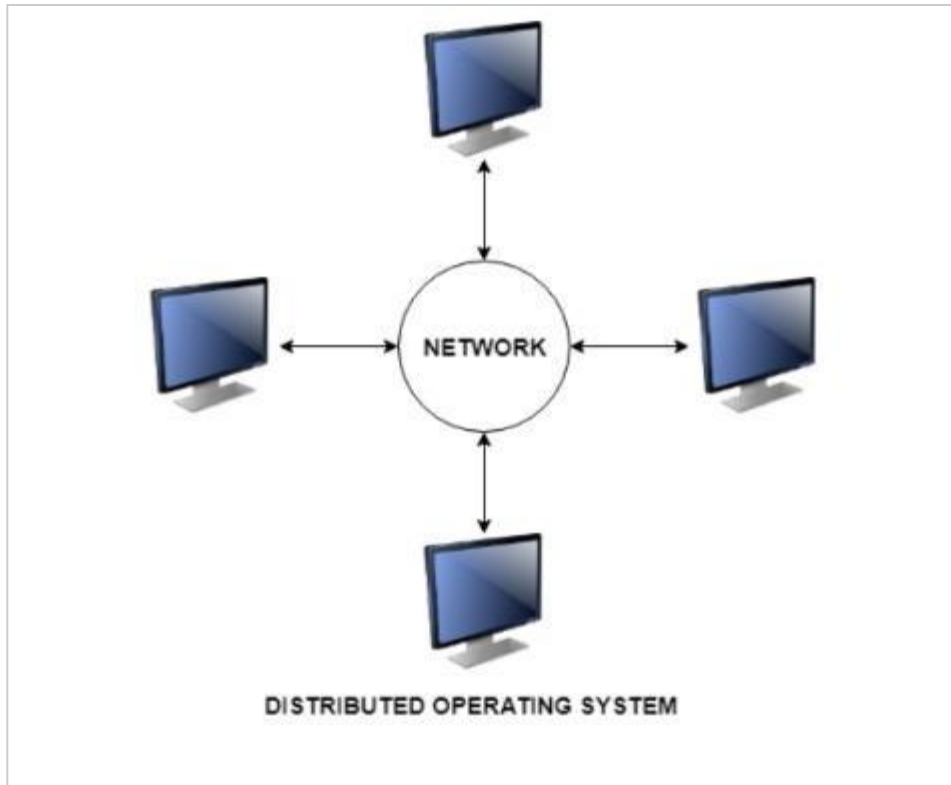
Berkeley's Algorithm

In this algorithm, the system chooses a node as master/ leader node. This is done from pool nodes in the server.

The algorithm is –

An election process chooses the master node in the server.
The leader then polls followers that provide their time in a way similar to Cristian's Algorithm, this is done periodically.
The leader then calculates the relative time that other nodes have to change or adjust to synchronize to the global clock time which is the average of times that are provided to the leader node.

Let's sum-up steps followed to synchronize the clock using the Berkeley algorithm,

Nodes in the distributed system with their clock timings –

N1 -> 14:00 (master node)

N2 -> 13: 46

N3 -> 14: 15

Step 1 – The Leader is elected, node N1 is the master in the system.

Step 2 – leader requests for time from all nodes.

N1 -> time : 14:00

N2 -> time : 13:46

N3 -> time : 14:20

Step 3 – The leader averages the times and sends the correction time back to the nodes.

N1 -> Corrected Time 14:02 (+2)

N2 -> Corrected Time 14:02 (+16)

N3 -> Corrected Time 14:02 (-18)

CONCLUSION:-

We have implemented the Berkeley algorithm for clock synchronization.

Questions:
1) Explain clock synchronization?

2) Explain Berkeley algorithm?

3) Describe advantages of Berkeley algorithm?

4) List applications of Berkeley algorithm.

Experiment No. 5

TITLE: - Token ring based mutual exclusion algorithm.

AIM: - Implement token ring based mutual exclusion algorithm.

THEORY: -

Token Ring algorithm achieves mutual exclusion in a distributed system by creating a bus network of processes. A logical ring is constructed with these processes and each process is assigned a position in the ring. Each process knows who is next in line after itself. When the ring is initialized, process 0 is given a token. The token circulates around the ring. When a process acquires the token from its neighbor, it checks to see if it is attempting to enter a critical region. If so, the process enters the region, does all the work it needs to, and leaves the region. After it has exited, it passes the token to the next process in the ring. It is not allowed to enter the critical region again using the same token. If a process is handed the token by its neighbor and is not interested in entering a critical region, it just passes the token along to the next process.

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 shared memory and a common physical clock.

(a)   (b)
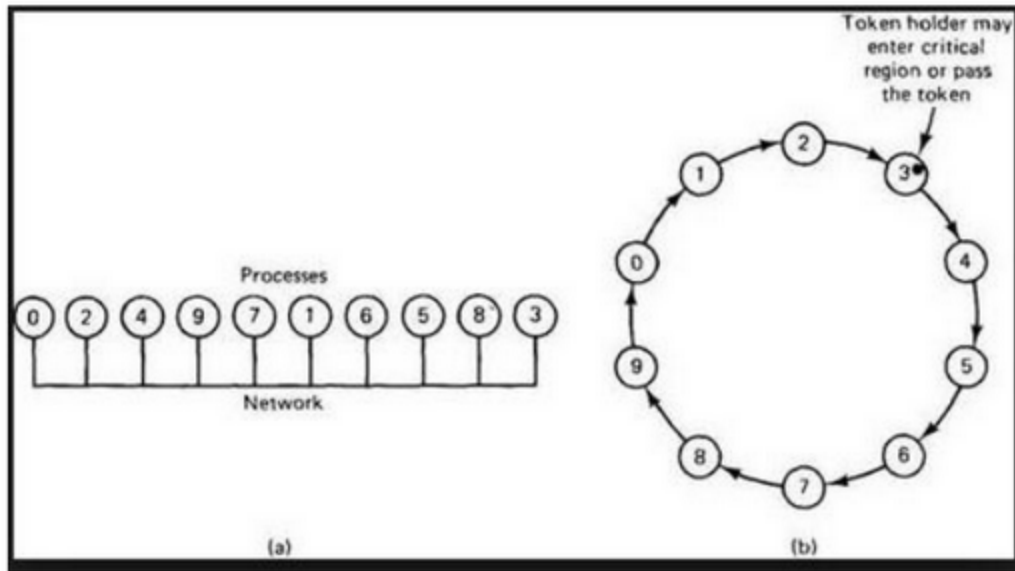
Advantages:

The correctness of this algorithm is evident. Only one process has the token at any instant, so only one process can be in a CS

Since the token circulates among processes in a well-defined order, starvation cannot occur.

Disadvantages

Once a process decides it wants to enter a CS, at worst it will have to wait for every other process to enter and leave one critical region.

If the token is ever lost, it must be regenerated. In fact, detecting that it is lost is difficult, since the amount of time between successive appearances of the token on the network is not constant. The fact that the token has not been spotted for an hour does not mean that it has been lost; some process may still be using it.

The algorithm also runs into trouble if a process crashes, but recovery is easier than in the other cases. If we require a process receiving the token to acknowledge receipt, a dead process will be detected when its neighbor tries to give it the token and fails. At that point the dead process can be removed from the group, and the token holder can pass the token to the next member down the line.

CONCLUSION:-

We have implemented a token ring based mutual exclusion algorithm.

Questions:

1) Describe mutual exclusion.
2) Describe a token ring.
3) List advantages of the token ring based mutual exclusion algorithm.
4) List Requirements of Mutual exclusion.

Experiment No. 6

TITLE: - Bully and Ring algorithm for leader election.

AIM: - Implement Bully and Ring algorithm for leader election.

THEORY: -

Distributed Algorithm is an algorithm that runs on a distributed system. Distributed system is a collection of independent computers that do not share their memory. Each processor has its own memory and they communicate via communication networks. Communication in networks is implemented in a process on one machine communicating with a process on another machine. Many algorithms used in the distributed system require a coordinator that performs functions needed by other processes in the system.

Election algorithms are designed to choose a coordinator.

Election Algorithms: Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected by another processor. Election algorithm basically determines where a new copy of the coordinator should be restarted. Election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has the highest priority number.Then this number is sent to every active process in the distributed system. We have two election algorithms for two different configurations of a distributed system.

The Bully Algorithm – This algorithm applies to systems where every process can send a message to every other process in the system. **Algorithm –** Suppose process P sends a message to the coordinator.

1. If the coordinator does not respond to it within a time interval T, then it is assumed that coordinator has failed.

2. Now process P sends election messages to every process with a high priority number.

3. It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.
4. Then it sends a message to all lower priority number processes that it is elected as their new coordinator.
5. However, if an answer is received within time T from any other process Q,
    ○ (I) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
    ○ (II) If Q doesn't respond within time interval T' then it is assumed to have failed and the algorithm is restarted.

Bully Algorithm

This synchronous algorithm assumes that each node has a unique ID and knows all the participant IDs.

The highest ID node declares itself the winner of the "election" by broadcasting a message to all the other nodes or lower ID's nodes. It then waits for a response before declaring itself the winner if they fail to respond.

An election is called when a high ID node is launched, the leader fails, or the heartbeat message fails.

2. The Ring Algorithm – This algorithm applies to systems organized as a ring(logically or physically). In this algorithm we assume that the link between the processes are unidirectional and every process can message to the process on its right only. Data structure that this algorithm uses is an active **list**, a list that has a priority number of all active processes in the system.

Algorithm –

1. If process P1 detects a coordinator failure, it creates a new active list which is empty initially. It sends election messages to its neighbor on the right and adds number 1 to its active list.

2. If process P2 receives message elect from processes on left, it responds in 3 ways:

   ○ (I) If the message received does not contain 1 in the active list then P1 adds 2 to its active list and forwards the message.

   ○ (II) If this is the first election message it has received or sent, P1 creates a new active list with numbers 1 and 2. It then sends election message 1 followed by 2.

   ○ (III) If Process P1 receives its own election message 1 then the active list for P1 now contains numbers of all the active processes in the system. Now Process P1 detects the highest priority number from the list and elects it as the new coordinator.

CONCLUSION:-

We have implemented the Bully and Ring algorithm for leader election.

Questions:
1) Describe a bully algorithm.
2) Describe a ring algorithm.
3) List advantages of Ring algorithm for leader election.

4) List advantages of the bully algorithm for leader election.

Experiment No. 7

TITLE: - Web service and write any distributed application to consume the web service.

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

THEORY: -

Web service is a standardized medium to propagate communication between the client and server applications on the WWW (World Wide Web). A web service is a software module that is designed to perform a certain set of tasks.

- Web services in cloud computing can be searched for over the network and can also be invoked accordingly.
- When invoked, the web service would be able to provide the functionality to the client, which invokes that web service.

Follow below simple steps to create and deploy simple Web Service and Web Service Client in Eclipse IDE.

Step-1

Install Apache Tomcat and add it to Eclipse in Server Tab – I'm using Tomcat version 9.0.10.

Step-2

Create a Dynamic Web Project (name: CrunchifyWS)

Step-3

Create java file under /src folder. Right Click /src folder -> New -> Class.

- Package: crunchify.com.web.service

- Name: CrunchifyHelloWorld.java

Step-4

Open CrunchifyHelloWorld.java file and create simple main method.

```
package crunchify.com.web.service

/**
```

```java
* @author Crunchify.com

*/

public class CrunchifyHelloWorld {

    public float addValue(float value) {

        return (value + 10);

    }

    public float subtractValue(float value) {

        return (value - 10);

    }

}
```

Step-5

- Right Click on file CrunchifyHelloWorld.java -> Web Services -> Create Web Service

- Select options as mentioned in below diagram.

- Click finish

Step-6

It may take some time to finish all processes and you should see new project "CrunchifyWSClient" created. Here is a final project structure:

▼ ⬚ > CrunchifyWS [eclipse-wp master]
  ▶ 🖥 JAX-WS Web Services
  ▶ 📋 Deployment Descriptor: CrunchifyWS
  ▼ 🗂 Java Resources
    ▼ 🗂 > src
      ▼ 📦 > crunchify.com.web.service
        ▶ 📄 CrunchifyHelloWorld.java
    ▶ 📚 Libraries
  ▶ 📚 JavaScript Resources
  ▶ 🗂 > build
  ▶ 🗂 > WebContent
▼ ⬚ > CrunchifyWSClient [eclipse-wp master]
  ▶ 🖥 JAX-WS Web Services
  ▶ 📋 Deployment Descriptor: CrunchifyWSClient
  ▼ 🗂 Java Resources
    ▼ 🗂 > src
      ▼ 📦 > crunchify.com.web.service
        ▶ 📄 CrunchifyHelloWorld.java
        ▶ 📄 CrunchifyHelloWorldProxy.java
        ▶ 📄 CrunchifyHelloWorldService.java
        ▶ 📄 CrunchifyHelloWorldServiceLocator.java
        ▶ 📄 CrunchifyHelloWorldSoapBindingStub.java
    ▶ 📚 Libraries
  ▶ 📚 JavaScript Resources
 ▶ 🗂 > build
  ▶ 🗂 > WebContent

Step-7

CrunchifyWS and CrunchifyWSClient both projects should be automatically deployed to server.

Also, Eclipse automatically opens Web Service Test Client Window with URL:
http://localhost:8080/CrunchifyWSClient/sampleCrunchifyHelloWorldProxy/TestClient.js
p?endpoint=http://localhost:5922/CrunchifyWS/services/CrunchifyHelloWorld

Step-8

Now click on addValue(float), subtractValue(float) and provide an input to check updated
result.
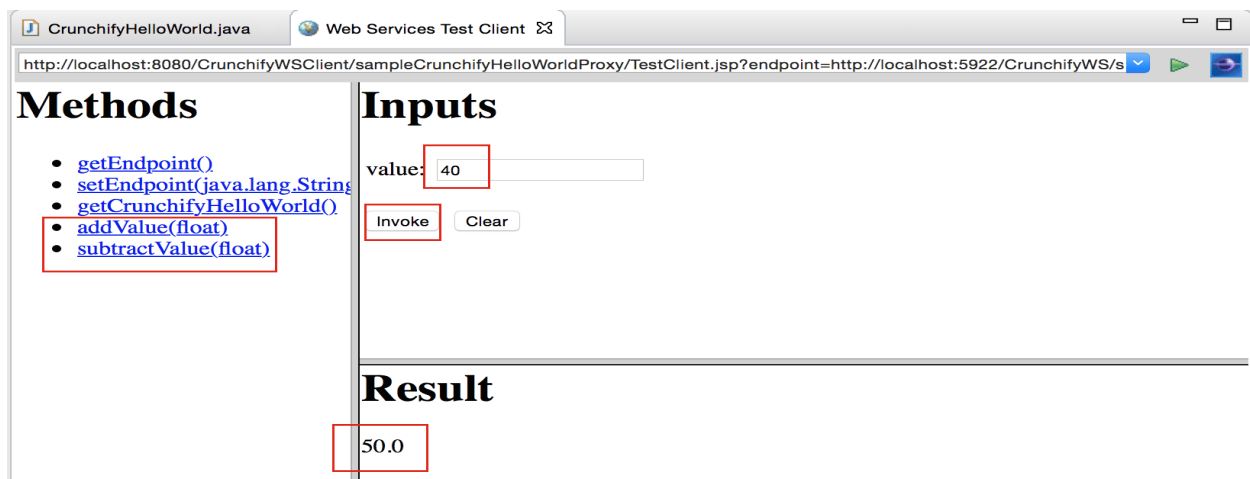
And you are all set. Do let me know if you see any difficulty with these steps.

Are you getting below error after clicking Invoke button?

Follow below steps:

1. Right click on Tomcat

2. Click Add and Remove to see application added

3. Click Clean…

4. Click Publish

5. Click Restart

1. Right click on Tomcat
2. Click Add and Remove to see application added
3. Click Clean...
4. Click Publish
5. Click Restart
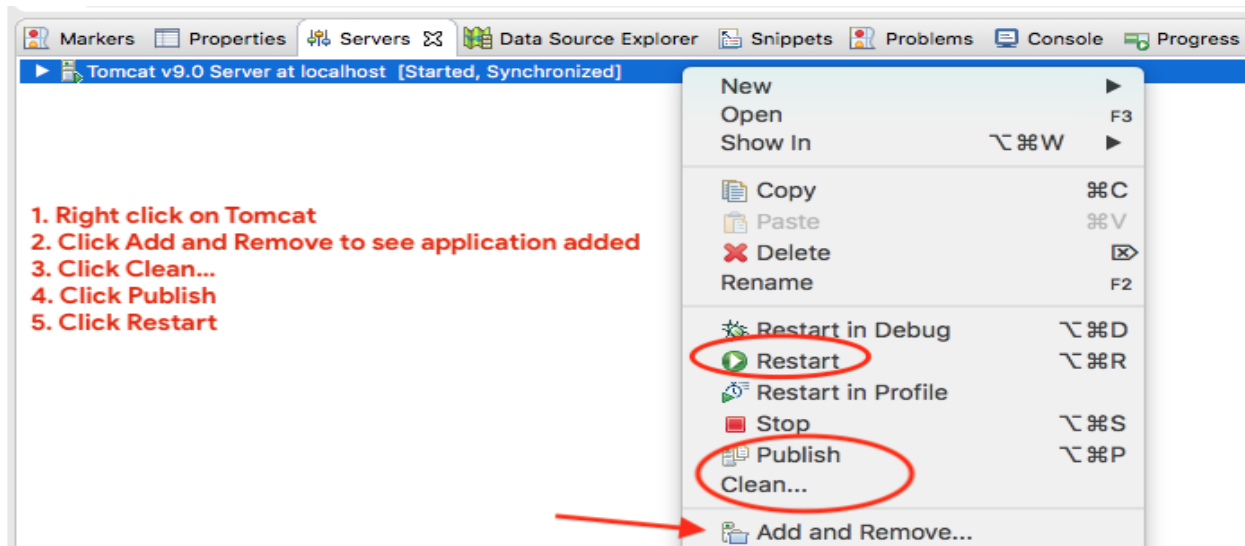
How to rerun WebService after restarting Application or later?
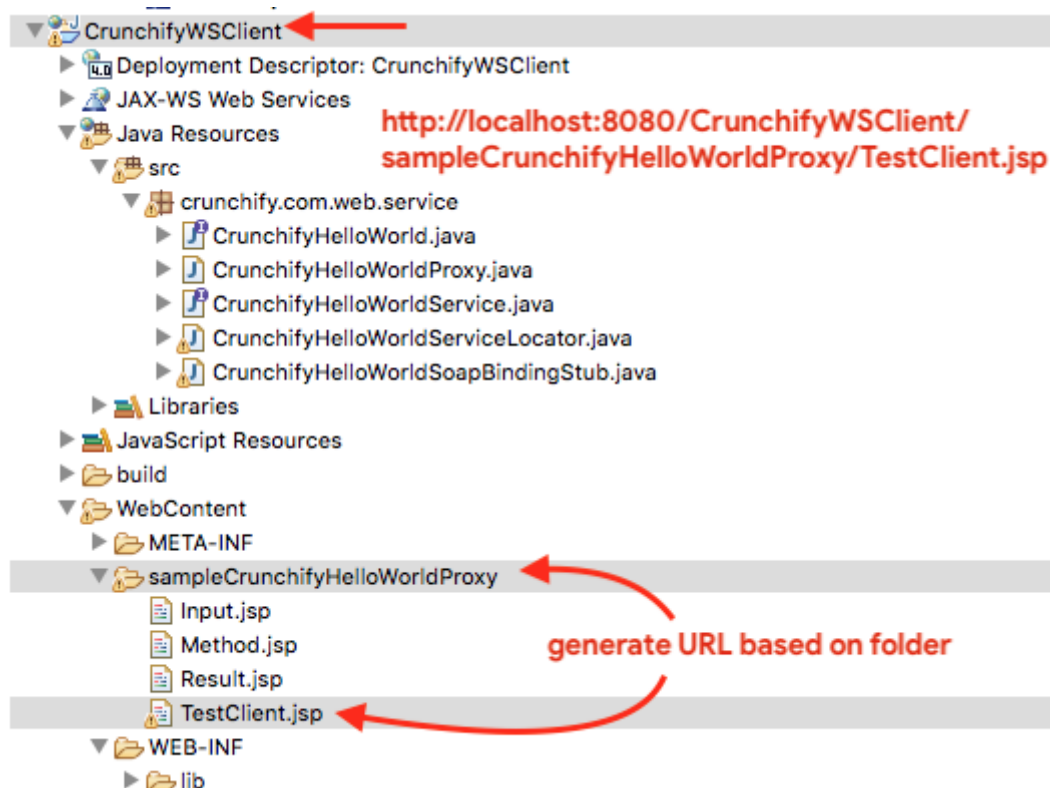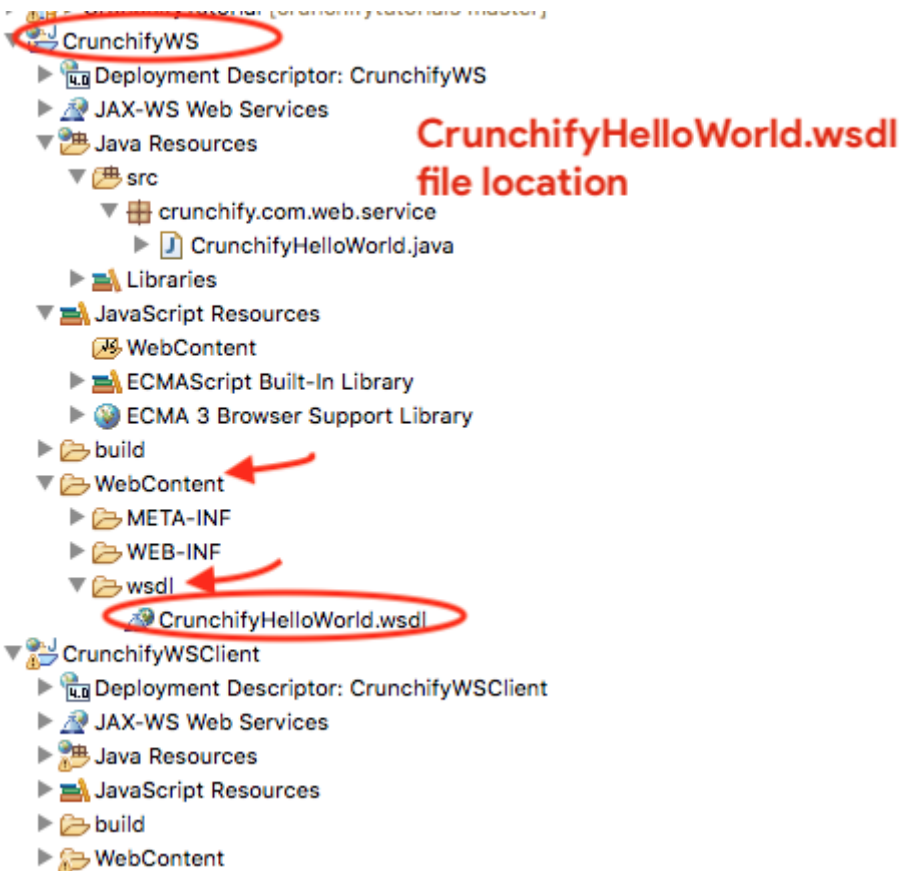
Here is a handy URL:
http://localhost:8080/CrunchifyWSClient/sampleCrunchifyHelloWorldProxy/TestClient.jsp

If you want to download CrunchifyHelloWorld.wsdl then here it is:

CrunchifyHelloWorld.wsdl file location

```xml
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions targetNamespace="http://service.web.com.crunchify"

xmlns:apachesoap="http://xml.apache.org/xml-soap"

xmlns:impl="http://service.web.com.crunchify"

xmlns:intf="http://service.web.com.crunchify"

xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!--WSDL created by Apache Axis version: 1.4

Built on Apr 22, 2006 (06:55:48 PDT)-->
```

```
<wsdl:types>

 <schema elementFormDefault="qualified"
targetNamespace="http://service.web.com.crunchify"
xmlns="http://www.w3.org/2001/XMLSchema">

 <element name="subtractValue">

  <complexType>

   <sequence>

    <element name="value" type="xsd:float"/>

   </sequence>

  </complexType>
```

CONCLUSION:-

We have implemented simple web service and write any distributed application to consume the web service.

Questions:
1) Define web service
2) List advantages of web service.
3) Describe a distributed application to consume the web service.
4) Summarize different types of web service.

Experiment No. 8

TITLE: - A Distributed Application for Interactive Multiplayer Games.

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

THEORY: -

Online multiplayer games have improved dramatically in the past few years. The newest online virtual worlds such as Everquest feature fantastic artwork, realistic graphics, imaginative gameplay and sophisticated artificial intelligence. Before the next generation of multiplayer games can complete the transition to lifelike virtual worlds, they must be able to support real-time interactions. The main obstacle to real-time interactions is the Internet's inability to provide low-latency guarantees. In this paper, we present a system for enabling real-time multiplayer games.

In particular, we focus on real-time multiplayer games that have strong consistency requirements. That is, all players must share a common view of a complex virtual world. The combination of low latency and absolute consistency is difficult to achieve because messages may be delayed indefinitely in the network. Our system realizes these goals using three building blocks: a Mirrored-Server architecture, a trailing state synchronization protocol, and a lowlatency reliable multicast protocol. We have implemented and integrated these three components into a working prototype and performed some preliminary experiments on our system.

We chose Quake as our proof-of-concept game because it is very sensitive to latency and provides a virtual world environment. Quake is a commercial game that was not designed for a distributed architecture. As such, Quake forces us to address the needs of a real game, as opposed to a game that fits neatly into our architecture. Another reason for using Quake is that it is one of the few commercial games for which the source code has been released.

Quake is a 3-D first player shooter, by id Software, the creators of a revolutionary and spectacularly successful line of games including Wolfenstein 3D, Doom, Quake and Quake III. Besides 3- D graphics, their major innovation was the online multiplayer game. College students found that they could put the newly installed dorm LAN's to use by engaging in mortal combat with their neighbors. Multiplayer Quake is a simple game. Your avatar has a gun and so does everyone else. You're going to die in the next minute or so, but before that

happens, your goal is to finish off as many of your opponents as possible. All interactions take place in real-time, and more than a 100 ms lay can be a crippling handicap.

Network Protocols Almost all of the messages exchanged between the client and the server during gameplay are sent unreliably. Because there are few reliable messages, the client and server can get away with an inefficient yet simple stop-and-wait protocol with single-bit sequence numbers. Although recovery of lost packets is not necessary, the server does require knowledge of which packets have been dropped so that it can implement the FEC scheme described above. This is accomplished using sequence numbers.

Distributed applications (distributed apps) are applications or software that run on multiple computers within a network at the same time and can be stored on servers or cloud computing platforms. Unlike traditional applications that run on a single system, distributed applications run on multiple systems simultaneously.

Distributed applications run on distributed computing systems, which are a collection of independent computers that appear to the user as a single system. Computers in a distributed system operate concurrently and fail independently. They're also asynchronous -- meaning there's no synchronization between their internal clocks. In addition to networks or cloud platforms, many distributed applications are also built and deployed on blockchain-based platforms.

Distributed apps are more resistant to certain cyber attacks because they have no single point of failure.

Networked games are rapidly evolving from small 4-8 person, one-time play games to large-scale games involving thousands of participants and persistent game worlds. However, like most Internet applications, current networked games are centralized. Players send control messages to a central server and the server sends (relevant) state updates to all other active players. This design suffers from the well known robustness and scalability problems of single server designs. For example, complex game-play and AI

computation prevent even well provisioned servers from supporting more than several tens of players for first person shooter (FPS) games. Further, client-server game designs often force players to rely on infrastructure provided by the game manufacturers. These infrastructures are sometimes not well provisioned or long-lived; thus, they either provide poor performance or prevent users from playing their game long after their purchase.

How do distributed apps work?

Distributed apps can communicate with multiple servers or devices on the same network from any geographical location. The distributed nature of the applications refers to data being spread out over more than one computer in a network.

Distributed applications are broken up into two separate models -- the client software and the server software. The client software or computer accesses the data from the server or cloud environment, while the server or cloud processes the data. Cloud computing can be used instead of servers or hardware to process a distributed application's data or programs. If a distributed application component goes down, it can <u>failover</u> to another component to continue running.

Types of distributed app architecture

Distributed applications need to be hosted on more than one server to run. They rely on distributed systems, which are a network of machines that send and receive information spread out in different locations and organized into an architecture. Distributed apps can be organized into the following categories based on their underlying network architecture:

- **Client-server.** In a client-server architecture, multiple clients, or workstations, request resources such as shared files or printers from servers over the internet. Clients don't provide resources to the server.

- **Service-oriented architecture**. In SOA, software components are made interoperable and reusable through service interfaces.

- **Microservices architecture.** A microservices architecture develops software systems in single-function modules and services, where each service is separately contained along with its data.

- **Peer-to-peer.** In a P2P architecture, there's no central server and each node in the network has the same responsibilities. Blockchain uses a P2P network architecture.

CONCLUSION:-

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

QUESTIONS:-

List types of Distributed Application.
Define Distributed Application.
How do distributed apps work?

Describe requirements of Interactive Multiplayer Games.

Experiment No. 9

TITLE: - Distributed application using Messaging System in Publish-Subscribe

AIM: - To develop any distributed application using Messaging System in Publish-Subscribe

THEORY: -

Introduction to Publish-Subscribe

Networking technologies and products now enable a high degree of connectivity across a large number of computers, applications, and users. In these environments, it is important to provide asynchronous communications for the class of distributed systems that operate in a loosely-coupled and autonomous fashion, and which require operational immunity from network failures. This requirement has been filled by various middleware products that are characterized as messaging, message oriented middleware (MOM), message queuing, or publish-subscribe.

Applications that communicate through a publish and subscribe paradigm require the sending applications (publishers) to publish messages without explicitly specifying recipients or having knowledge of intended recipients. Similarly, receiving applications (subscribers) must receive only those messages that the subscriber has registered an interest in.

This decoupling between senders and recipients is usually accomplished by an intervening entity between the publisher and the subscriber, which serves as a level of indirection. This intervening entity is a queue that represents a subject or channel

Each queue manager matches messages published to a topic with the locally created subscriptions that have subscribed to that topic. You can configure a network of queue managers so that messages published by an application connected to one queue manager are delivered to matching subscriptions created on other queue managers in the network. This requires additional configuration over simple channels between queue managers.

A distributed publish/subscribe configuration is a set of queue managers connected together. The queue managers can all be on the same physical system, or they can be distributed over several physical systems. When you connect queue managers together, subscribers can subscribe to one queue manager and receive messages that were initially published to another queue manager. To illustrate this, the following figure adds a second

queue manager to the configuration described in Example of a single queue manager publish/subscribe configuration.

- Queue manager 2 is used by Publisher 4 to publish weather forecast information, using a topic of Weather, and information about traffic conditions on major roads, using a topic of Traffic.
- Subscriber 4 also uses this queue manager, and subscribes to information about traffic conditions using topic Traffic.
- Subscriber 3 also subscribes to information about weather conditions, even though it uses a different queue manager from the publisher. This is possible because the queue managers are linked to each other.
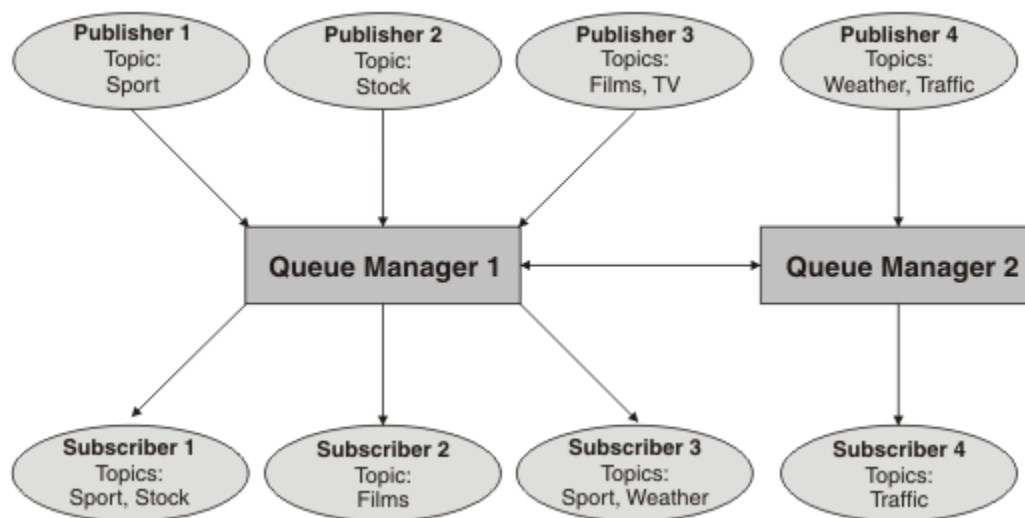
More and more of the web is moving to microservice architecture, which allows for loosely-coupled services to work together to provide functionality to users. While these services will often communicate with each other via network requests, the publish-subscribe model is another common way for this collaboration to occur.

There are many different technologies that can be used to implement pub-sub, including but not limited to Apache Kafka, Amazon Kinesis, Google Pub/Sub, and Microsoft Event

Hubs. The purpose of this post is not to go into the details of implementation or discuss the pros and cons of these specific technologies, but to provide definitions of the words and an overview of how it works.

I talk about two different types of events that can be passed through a pub-sub model, Change Data Capture events and "business events" here, though these messages could of course be anything!

What is decoupling and why is it good?

Before we dive into pub-sub, let's take a quick detour to decoupling. If we're not convinced of the value of decoupling, pub-sub won't do us any good!

Decoupling means reducing the dependency between pieces of a system. This can mean two classes in the same codebase or two services in a system built on microservices, or anything in between.

Ok, but what do we mean by "reducing the dependency"? Obviously the different parts of the system rely on each other — they're part of the same system! Typically what people mean when they talk about the level of dependency between systems is whether or not one part of the system needs to be changed when the other does, or whether they can be updated independently of the other.

Pub-sub is decoupled because using it, we can change anything we want about a publisher — we could even go so far as to rebuild it using an entirely different language or framework! — but as long as it continues to produce messages in the same way, we don't need to change anything at all about the downstream subscriber. Same goes in reverse — as long as a subscriber continues to be able to process messages that come to it in the same format, we can change anything we want about it and the publisher doesn't need to be aware of this change at all.

This will make more sense once we dive into this a bit more, so let's do that!
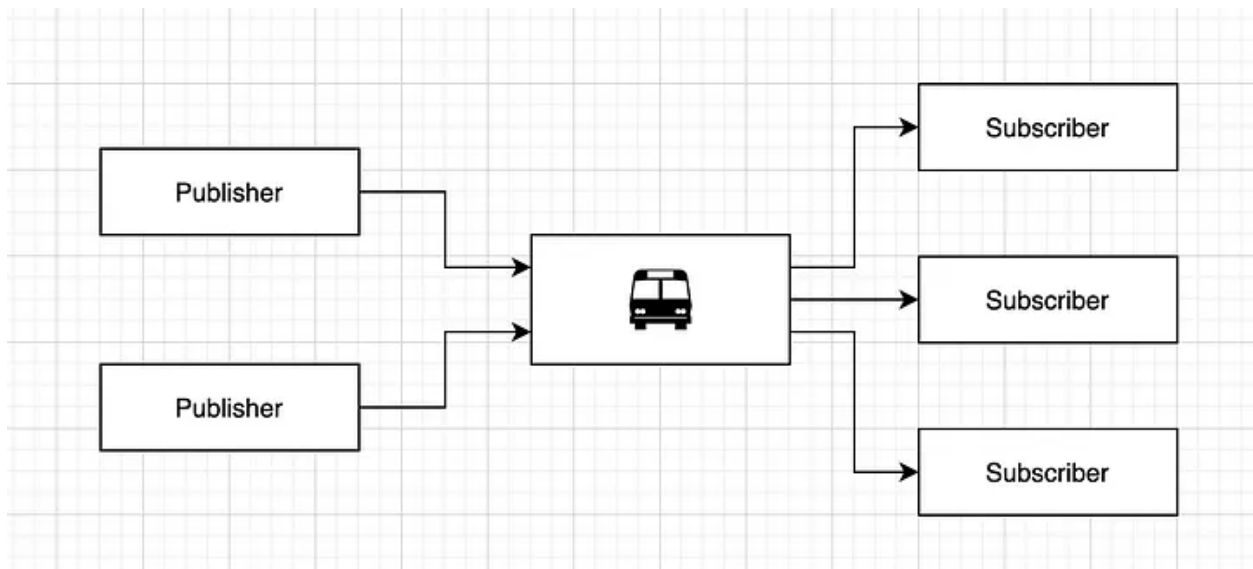
Publishing

This also sometimes referred to as "producing." A publisher is anything that puts messages onto the message bus. This can be a database trigger, a web request, an API call, running a

script, whatever. Messages typically take a pre-defined format, and are in JSON. They are pushed to a specific topic.

Subscribing

This also sometimes referred to as "consuming." Subscribers are typically responsible for a specific piece of behavior, and will subscribe to a specific topic. The beauty of pub-sub is that you can have as many subscribers as you want, which allows you to scale them all separately, depending on how long each message takes a given subscriber to process, and how much traffic is on the given topic to which they subscribe.

Overview of publish/subscribe clusters

A publish/subscribe cluster is a standard cluster with one or more topic objects added to the cluster. When you define an administrative topic object on any queue manager in a cluster, and make that topic object clustered by specifying a cluster name, then publishers and subscribers to the topic can connect to any of the queue managers in the cluster, and messages published are routed to the subscribers over cluster channels between queue managers.

CONCLUSION:-
We have implemented Distributed application using Messaging System in publish-Subscribe

QUESTIONS:-

1.   Define publish-Subscribe
2.   Describe decoupling and why is it good?
3.   List examples of Publish-Subscribe Mechanism
4.   Discuss advantages of Publish-Subscribe.