DISTRIBUTED SYSTEMS
Semester VIII

Assignment 1

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

What is Multithreading?
Multithreading is a programming technique that allows a program to execute multiple tasks concurrently within a single process.
Instead of running tasks sequentially, multithreading divides a program's work into smaller, independent units called threads, which can run simultaneously.
This can improve program performance and responsiveness, especially for tasks that involve waiting or I/O
Thread States:

1. **Running:** The thread is currently executing.
2. **Dead:** The thread has completely finished executing, and will never execute again. This typically means that the client call to the server has completed.
3. **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.
4. **Blocked:** A blocked thread is waiting for some external event to finish.
5. **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.
6. **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.

Ex, a blocked 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.

What is RMI?

Remote Method Invocation (RMI) is an API that allows an object in a JVM present on Client-side to invoke methods on an object present in another JVM on Server-side.
RMI creates a public remote server object that enables client and server-side communications through simple method calls on the server object.

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:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote JVM
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally returns the value to the caller.

The stub object on the client machine builds an information block and sends this information to the server.
The block consists of

- An identifier of the remote object to be used
- Method name which is to be invoked
- Parameters to the remote JVM

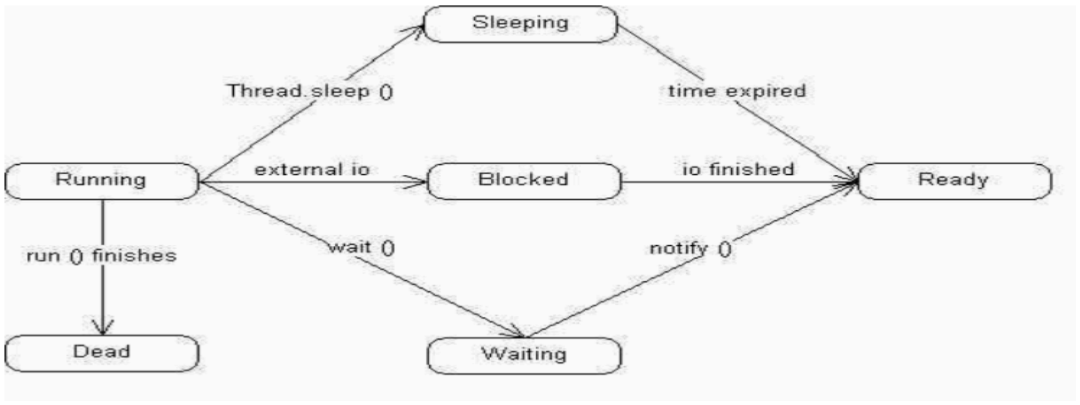
Skeleton

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 from the stub, it passes the request to the remote object. It performs the following tasks

- It calls the desired method on the real object present on the server.
- It forwards the parameters received from the stub object to the method.
- It writes and transmits (marshals) the result to the caller

Steps to write the RMI program

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the RMIC tool
4. Start the registry service by RMIRegistry tool
5. Create and start the remote application
6. Create and start the client application



Object Synchronization and Data Synchronization

Assignment 2

Develop any distributed application using CORBA to demonstrate object brokering

What is CORBA?

CORBA (Common Object Request Broker Architecture) is a middleware standard developed by the Object Management Group (OMG) that facilitates communication between distributed objects in a network. It enables programs written in different languages and running on various platforms to interact with each other as if they were local objects.

Key Components of CORBA:

1. **Object Request Broker (ORB):**
    - The core of CORBA architecture that handles communication between client and server objects.
    - It locates objects, sends requests, and manages data exchange between systems.
2. **Interface Definition Language (IDL):**
    - Defines the interfaces that objects expose to clients, ensuring language-independent communication.
    - Developers write the interface in IDL, and it is compiled into specific language stubs and skeletons.
    - IDL object interfaces describe, among other things:
        i. The data that the object makes public.
        ii. 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 data types.
        iii. Exceptions that the object can throw. IDL exceptions are also mapped to Java exceptions, and the mapping is very direct.
3. **Client:**
    - The entity that requests a service from a remote object.
    - Uses stubs (auto-generated code) to communicate with the ORB.
4. **Server (Object Implementation):**

    - Provides the actual implementation of the remote methods.
    - Contains skeleton code that interacts with the ORB.

How CORBA Works:

1. The client calls a method on a remote object through the stub.
2. The ORB marshals (converts) the request and sends it to the server.
3. The server ORB unmarshals the request and invokes the method on the object.
4. The result is marshaled and sent back to the client.

Steps involved in developing CORBA applications:

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

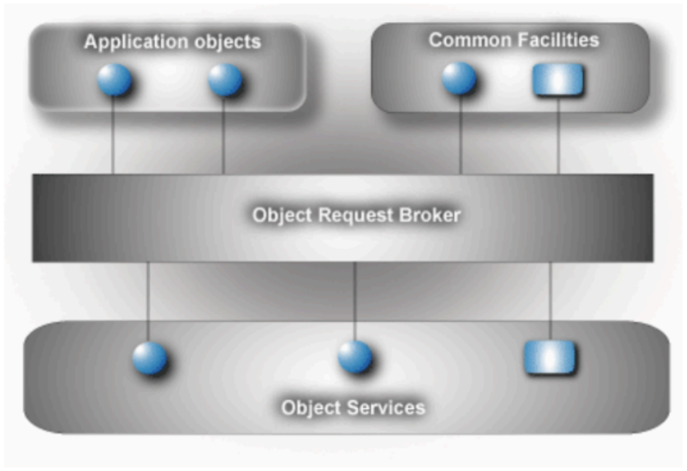
Internet Inter-ORB Protocol (IIOP)

CORBA specifies this network protocol. It 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.

Essential building blocks of an Oracle8i JServer CORBA application

1. **ORB:** how to talk to remote objects
2. **IDL:** how to write a portable interface
3. **The naming service (and JNDI):** how to locate a persistent object
4. **Object adapters:** how to register a transient object



The basic steps for CORBA development include: (Lab Manual)

Assignment 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.

Message Passing Interface (MPI)

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 parallelizing tasks very easy.

It is available in the form of a standard library.

The parallelism between different processes can be achieved with the 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.

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. Syntax:

```
int MPI_Send ( void *data_to_send,
               int send_count,
               MPI_Datatype send_type,
               int destination_ID,
               int tag,
               MPI_Comm comm);
```

- **MPI_Recv**: to receive a message from another process. Syntax:

```
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.

MPI Programming:

1. Installing MPI Library: 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.
2. Include the mpi library in you C/C++ program by “#include<mpi.h>”.

Some of the MPI library functions and variables:

- **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 an MPI Program?

# mpicc -o objectname programname.c

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

To run the program with the “mpirun” command. You can run literally on any number of processes.

# mpirun -np total_process_number ./objectname

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

Assignment 4

Implement Berkeley algorithm for clock synchronization

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:

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

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.

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.

The algorithm is −

1. An election process chooses the master node in the server.
2. The leader then polls followers that provide their time in a way similar to Cristian’s Algorithm, this is done periodically.
3. 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.

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)

Assignment 5

Implement token ring based mutual exclusion algorithm

1. Token Ring algorithm achieves mutual exclusion in a distributed system by creating a bus network of processes.
2. A logical ring is constructed with these processes and each process is assigned a position in the ring.
3. Each process knows who is next in line after itself.
4. When the ring is initialized, process 0 is given a token. The token circulates around the ring.
5. 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.
6. After it has exited, it passes the token to the next process in the ring.
7. It is not allowed to enter the critical region again using the same token.

8. 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 ensures concurrency control 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.

Advantages:

1. The correctness of this algorithm is evident.
2. Since the token circulates among processes in a well-defined order, starvation cannot occur.
3. 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.

Disadvantages:

1. 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.
2. 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.

Assignment 6

Implement Bully and Ring algorithm for leader election

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 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 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. Then this number is sent to every active process in the 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.

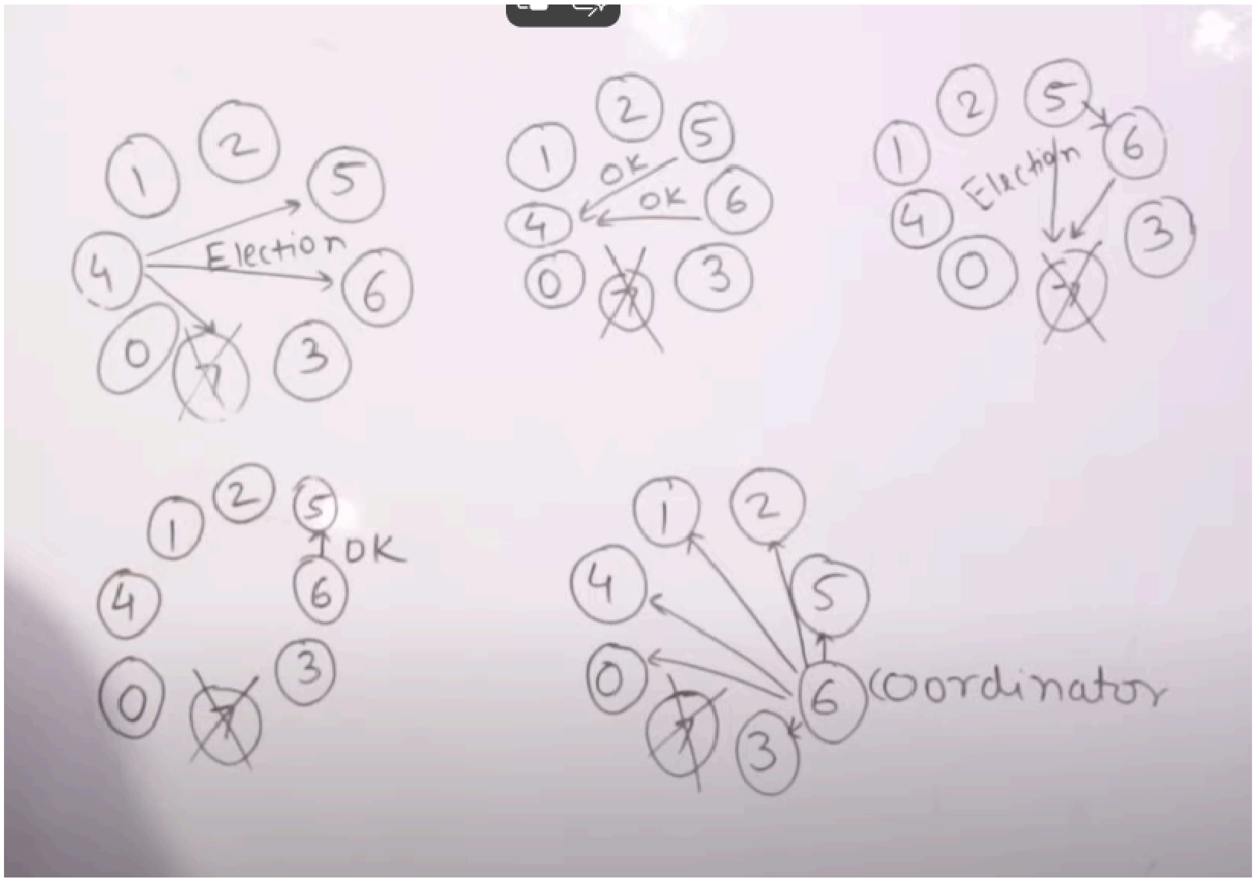
1. If the coordinator does not respond to it within a time interval T, then it is assumed that coordinator has failed.
2. Now process P sends election messages to every process with a high priority number.
3. It waits for responses, if no one responds for time interval T then process P elects itself 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,
    1. Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
    2. If Q doesn't respond within the time interval T' then it is assumed to have failed and the algorithm is restarted.

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**.**

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:
    a. If the message received does not contain 1 in the active list then P1 adds 2 to its active list and forwards the message.
    b. 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.
    c. 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.

Assignment 7

What is a web-service?

A web service is a software system designed to support interoperable machine-to-machine interaction over a network. Distributed applications can consume web services by creating a client that interacts with the service's interface, often using protocols like HTTP or HTTPS and data formats like XML or JSON