Advanced Operating Systems
Programming Project 3- Design Documention

Name:Manisha Nalla **R11471258**

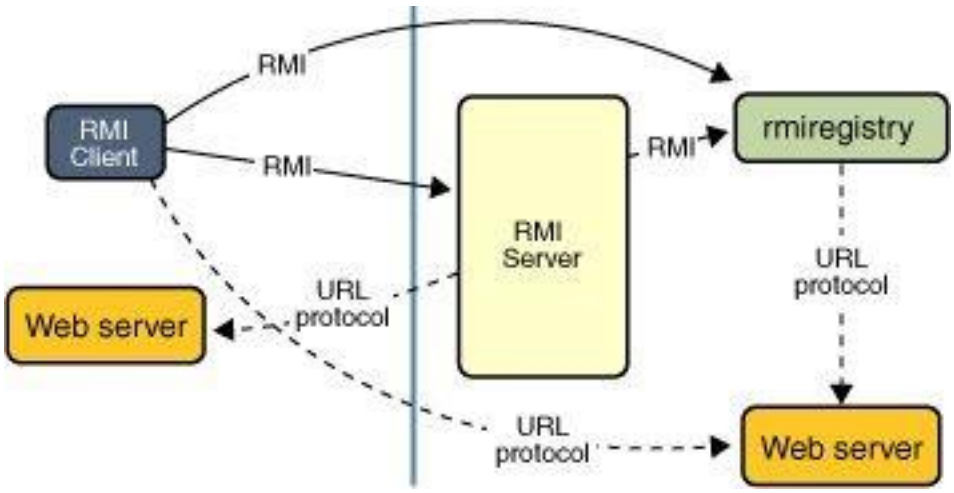
---

1. Java RMI

The Java Remote Method Invocation (RMI) system allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine. RMI provides for remote communication between programs written in the Java programming language.

RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a *distributed object application*.

The following illustration depicts an RMI distributed application that uses the RMI registry to obtain a reference to a remote object. The server calls the registry to associate (or bind) a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it. The illustration also shows that the RMI system uses an existing web server to load class definitions, from server to client and from client to server, for objects when needed.



a)How to Create Remote Interfaces,Objects and Methods ?

Like any other Java application, a distributed application built by using Java RMI is made up of interfaces and classes. The interfaces declare methods. The classes implement the methods declared in the interfaces and, perhaps, declare additional methods as well. In a distributed application, some implementations might reside in some Java virtual machines but not others. Objects with methods that can be invoked across Java virtual machines are called *remote objects*.

An object becomes remote by implementing a *remote interface*, which has the following characteristics:

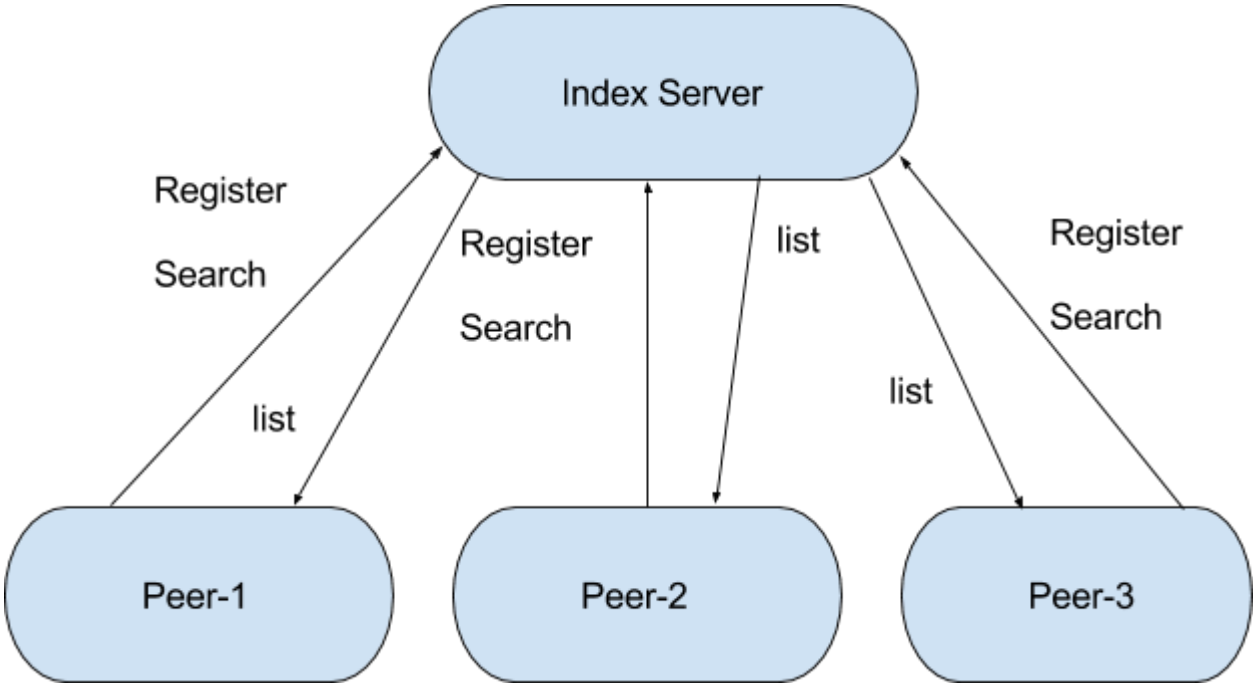
- A remote interface extends the interface `java.rmi.Remote`.
- Each method of the interface declares `java.rmi.RemoteException` in its `throws` clause, in addition to any application-specific exceptions.

2.Program Design:

a)Central Indexing Server: Central indexing server indexes the contents of all the peers that register with it and search facility to peers.

- Each peer register the files of shared directory using registry method with peer id and filename as parameters.when peer invokes register method all its files will be registered by indexing server with index
- Each peer can search files when connected to index server and returns all the matching peers to the requestor.

b)Peer: In Napster-style peer-to-peer file sharing system peer acts as both client and server. When a peer search for file using the search method then indexing server returns the list of the other peers that hold the file . the user can pick one such peer and the client then connects to this peers and sends the requested file.



3.Implementation:

NapsterServer:- This creates Indexing server having following remote methods

Insertintoregistry: insert peerId and filename into central indexing server
Peersearch: search file name and returns peerid
Insertportnumber: insert port number and peerid
Returnportnumber: takes peerid and returns portnumber

ThreadMainClass:- It creates two threads one for client and other for server

ThreadClient:- This creates a client which connected to Index server & register files and then searches the file and download the file by connecting to other peers

Threadserver:- This creates Server for clients and have following remote methods

Obtain: downloads the file into the shared directory
SetDirectory: sets client and server directory to send data.

ServerInterface: This interface consists remote methods to be accessed by peer.

ServerClass: this class implements serverInterface and override abstract methods so that do have some functionality when they are called .

4.Execution Steps:

Step1: First comple all the files using the command

make

Execute Central server using the following commands

java CentralServer

Step2: execute each peer(MainThread) by giving port number for each peer for arg[0]

Eg: java MainThread 8022

5. Imporvements:

A) Displaying file if big cannot be done.
B) We need to set directory to obtain file.