MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGY

Banjara Hills, Hyderabad, Telangana

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Distributed System Laboratory Manual

Academic Year 2016-2017

Table of Contents

I Contents

II Programs

Part I

Contents

1. Vision of the Institution

To be part of universal human quest for development and progress by contributing high calibre, ethical and socially responsible engineers who meet the global challenge of building modern society in harmony with nature.

2. Mission of the Institution

- To attain excellence in imparting technical education from undergraduate through doctorate levels by adopting coherent and judiciously coordinated curricular and co-curricular programs.
- To foster partnership with industry and government agencies through collaborative research and consultancy.
- To nurture and strengthen auxiliary soft skills for overall development and improved employability in a multi-cultural work space.
- To develop scientific temper and spirit of enquiry in order to harness the latent innovative talents.
- To develop constructive attitude in students towards the task of nation building and empower them to become future leaders
- To nourish the entrepreneurial instincts of the students and hone their business acumen.
- To involve the students and the faculty in solving local community problems through economical and sustainable solutions.

3. Department Vision

To contribute competent computer science professionals to the global talent pool to meet the constantly evolving societal needs.

4. Department Mission

Mentoring students towards a successful professional career in a global environment through quality education and soft skills in order to meet the evolving societal needs.

5. Programme Education Objectives

1. Graduates will demonstrate technical skills and leadership in their chosen fields of employment by solving real time problems using current techniques and tools.

2. Graduates will communicate effectively as individuals or team members and be successful in the local and global cross cultural working environment.

3. Graduates will demonstrate lifelong learning through continuing education and professional development.

4. Graduates will be successful in providing viable and sustainable solutions within societal, professional, environmental and ethical contexts

6. Programme Outcomes

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. **Lifelong learning:** Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

7. Programme Specific Outcomes

The graduates will be able to:

PSO1: Demonstrate understanding of the principles and working of the hardware and software aspects of computer systems.

PSO2: Use professional engineering practices, strategies and tactics for the development, operation and maintenance of software

PSO3: Provide effective and efficient real time solutions using acquired knowledge in various domains.

Part II

Programs

Program 1

FTP CLIENT USING GUI

Problem Definition

Write a program to create FTP client using GUI

Problem Description

A FTP client is to be created that can contact the FTP server to get services form it on behalf of the user.

A FTP client has to first contact the ftp server and provide the credentials.

In upload operation a file from the computer on which client is running is transferred to the ftp server, here the name and path of both source and destination files has to be given.

In download operation a file from the ftp server is transferred to the computer on which client is running, here also the name and path of both source and destination files has to be given.

In listing operation, the contents of the ftp server as listed, same as dir/ls on local machine.

Steps: Downloading

1. Write the code to create and place appropriate GUI components on the dailog box.
2. Connect or ftp server by providing the user name and password.
3. Execute 'get' command on ftp server.

Steps: Uploading

1. Write the code to create and place appropriate GUI components on the dailog box.
2. Connect or ftp server by providing the username and password.
3. Execute 'put' command on ftp server.

Steps: Listing

1. Write the code to create and place appropriate GUI components on the dailog box.
2. Connect or ftp server by providing the username and password.
3. Execute 'lst'command on ftp server.

The server process should work in connection -oriented and concurrent-server mode. You need to first start the server process in a server host and publish its host name and port number. A user on another host can then issue a command like

%myftp server−host−name server−port−number

to download a file from or upload a file to the server. After accepting the above myftp command, the client process should responds with prompt ftp>, waiting for user's ftp commands.

Requirements:

1. myftp is so named as to tell from the standard ftp command.
2. You need to consider the following ftp commands: ftp > put filename : to upload a file named filename to the server
   ftp > get filename : to download a file named filename from the server
   ftp > ls : to list the files under the present directory of the server
   ftp > cd : to change the present working directory of the server
   ftp > pwd : to display the present working directory of the server
   ftp>!ls : to list the files under the present directory of the client
   ftp>!cd : to change the present directory of the client
   ftp>!pwd : to display the present working directory of the client
   ftp>quit : to quit from ftp session and return to Unix prompt.

Pseudocode for Client Appl:

```
sd=socket(AF_INET, SOCK_STREAM, 0)
get server host name from argv[1] and server port number from argv[2]
connect(sd, &server-socket-address,...)

While (1) {
show ftp>
fgets a ftp command line from keyboard

if the command is "put a existed file"
0. send the command to the server
1. open the file
2. read the file
3. write the file to server
4. close the file


if the command is "put a non-existed file" display "filename: no such
file on client"
```

```
if the command is "get a file"
1. send the command to the server
2. fgets first line from the server:existed or nonexisted
3. if existed
3.1 read the file from the server
3.2 write the file to the local directory
4. if nonexisted
display "filename: no such file on server"


if the command is "cd ..." , "ls ...", or \pwd"
1.    send the command to the server
2.    fgets a reply line from the socket to see if the command is
      successfully executed
3.    read from the socket and display correspondingly.


if the command is "!ls ..." or \!pwd"
1. call system(command) locally


if the command is    "!cd    directory"
1. call chdir (directory) locally. Note that system( ) cannot
   execute "cd ..."


if the command is "quit"
1. close the socket
2. break or exit


otherwise: show "An invalid ftp command."
}
```

Algorithm for FTP Server:

Server:

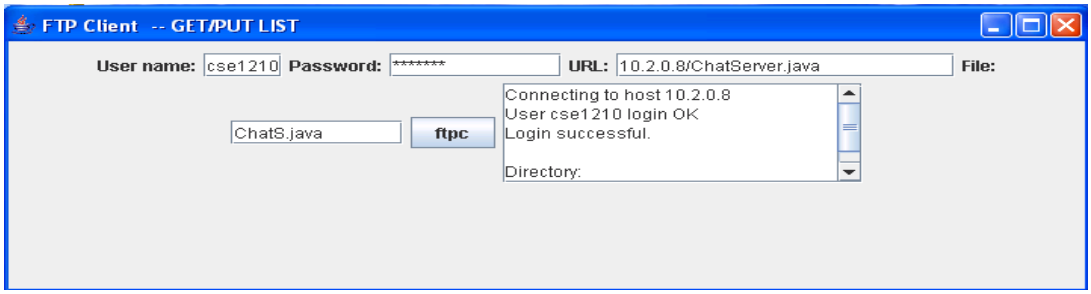
1. Include the necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3. Initialize server address to 0 using the bzero function.
4. Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
5. Bind the local host address to socket using the bind function.
6. Listen on the socket for connection request from the client.
7. Accept connection request from the Client using accept function.
8. Within an infinite loop, receive the file name from the Client.
9. Open the file, read the file contents to a buffer and send the buffer to the Client.

Client:

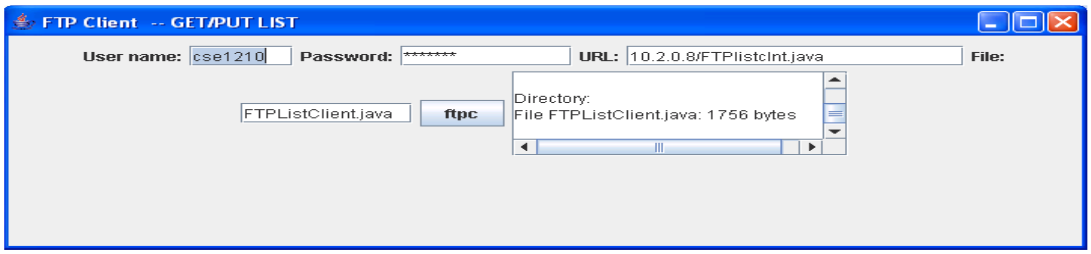
1. Include the necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3. Initialize server address to 0 using the bzero function.
4. Assign the sin_family to AF_INET.
5. Get the server IP address and the Port number from the console.
6. Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
7. Within an infinite loop, send the name of the file to be viewed to the Server.
8. Receive the file contents, store it in a file and print it on the console.

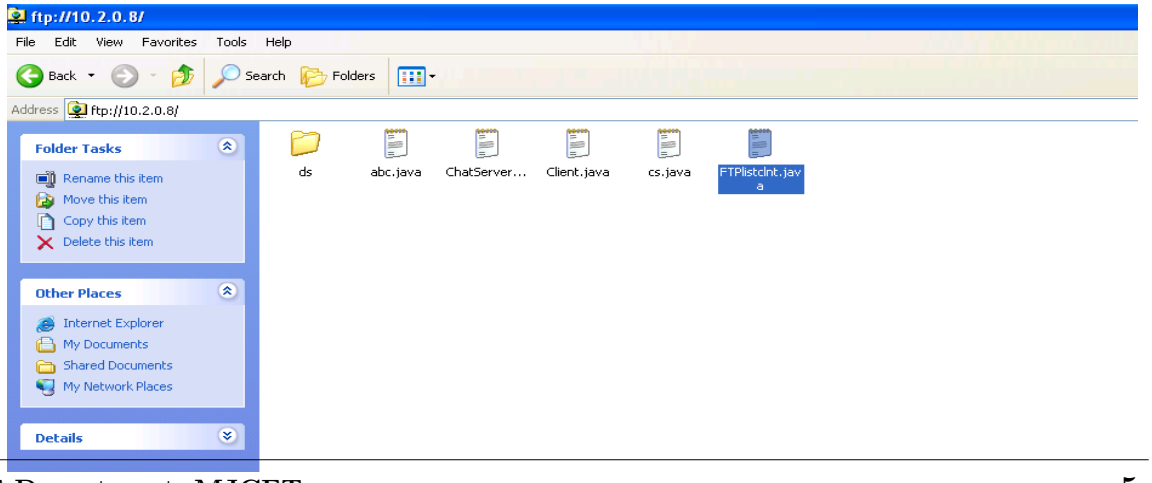
Input & Output Screen Shots:

GET



PUT





Program 2

CREATE CHAT APPLICATION

Problem Definition

Write a program to create chat application

Problem Description

Using TCP sockets connection has to be established, then messages can be send in full duplex manner until the connection is closed.

Differences between UDP and TCP

TCP is connection oriented, and provides error and flow control. UDP provides no such services, and relies on the application layer for them. UDP allows sending a packet with or without checksum; no connection is maintained, so each packet is sent independently. If a packet gets lost or the packets arrive out of order, then the application should detect and remedy the situation on its own. Also, UDP doesn′t give the security features of TCP like the three-way handshake.

UDP supports multicast sending a single packet to multiple machines. This is useful as it saves bandwidth, each packet is transmitted only once and the entire network receives it. UDP is also used in places where the overhead (delay) involved with TCP is expensive.

Some applications of UDP are in VoIP, streaming of audio and video, DNS, TFTP, SNMP, online gaming, and etcetera.

Given Requirements

There are two hosts, Client and Server. The Client accepts the message from the user and sends it to the Server. The Server receives the message, prints it and echoes the message back to the Client.

Technical Objective

To implement an UDP Echo Client-Server application , where the Client on establishing a connection with the Server, sends a string to the Server. The Server reads the String, prints it and echoes it back to the Client.

Algorithm

Server:

1. Include the necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
3. Initialize server address to 0 using the bzero function.
4. Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to SERVER_PORT, a macro defined port number.
5. Bind the local host address to socket using the bind function.
6. Within an infinite loop, receive message from the client using recvfrom function, print it on the console and send (echo) the message back to the client using sendto function.

Client:

1. Include the necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_DGRAM.
3. Initialize server address to 0 using the bzero function.
4. Assign the sin_family to AF_INET.
5. Get the server IP address and the Port number from the console.
6. Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
7. Within an infinite loop, read message from the console and send the message to the server using the sendto function.
8. Receive the echo message using the recvfrom function and print it on the console.

Sample Output

Server:
(Host Name:Root1)

```
[root@localhost 4ita33]# vi udpserver.c
[root@localhost 4ita33]# cc udpserver.c
[root@localhost 4ita33]# ./a.out
Server is Running...
Message is received
Send data to UDP Client: hi
Message is received
Send data to UDP Client: how are u
```

Client:
(Host Name:Root2)

```
[root@localhost 4ita33]# vi udpclient.c
[root@localhost 4ita33]# cc udpclient.c
[root@localhost 4ita33]# ./a.out 127.0.0.1
Enter input data :
hi
Data sent to UDP Server:hi
Received Data from server: hi

Enter input data :
how are u
Data sent to UDP Server:how are u
Received Data from server: how are u
```

INFERENCE:

Thus, the UDP ECHO client server communication is established by sending the message from the client to the server and server prints it and echoes the message back to the client.

Program 3

CREATE CHAT SERVER

Client/Server Communication

At a basic level, network-based systems consist of a server, client, and a media for communication as shown in below figure. A computer running a program that makes a request for services is called client machine. A computer running a program that offers requested services from one or more clients is called server machine. The media for communication can be wired or wireless network.

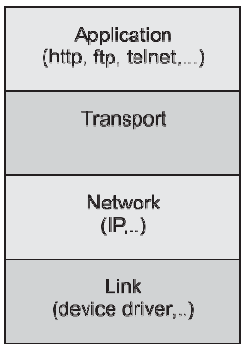


Fig. 13.2 TCP/IP software stack

Generally, programs running on client machines make requests to a program (often called as server program) running on a server machine. They involve networking services provided by the transport layer, which is part of the Internet software stack, often called *TCP/IP (Transport Control Protocol/Internet Protocol) stack*, shown in Fig. 13.2. The transport layer comprises two types of protocols, *TCP (Transport Control Protocol)* and *UDP (User Datagram Protocol)*. The most widely used programming interfaces for these protocols are sockets.

TCP is a connection-oriented protocol that provides a reliable ow of data between two computers. Example applications that use such services are HTTP, FTP, and Telnet.

UDP is a protocol that sends independent packets of data, called *datagrams*, from one computer to another with no guarantees about arrival and sequencing. Example applications that use such services include Clock server and Ping.

The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer. Port is represented by a positive (16-bit) integer value. Some ports have been reserved to support common/well known services:

- ftp 21/tcp
- telnet 23/tcp
- smtp 25/tcp
- login 513/tcp
- http 80/tcp,udp
- https 443/tcp,udp

User-level process/services generally use port number value >= 1024.

Here we are creating a chat server using TCP

Given Requirements

TThere are two hosts, Client and Server. Both the Client and the Server exchange message i.e. they send messages to and receive message from the other. There is a two way communication between them.

Technical Objective

To implement a full duplex application, where the Client establishes a connection with the Server. The Client and Server can send as well as receive messages at the same time. Both the Client and Server exchange messages.

Algorithm

Server:

1. Include the necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3. Initialize server address to 0 using the bzero function.
4. Assign the sin_family to AF_INET, sin_addr to INADDR_ANY, sin_port to dynamically assigned port number.
5. Bind the local host address to socket using the bind function.
6. Listen on the socket for connection request from the client.
7. Accept connection request from the Client using accept function.
8. Fork the process to receive message from the client and print it on the console.
9. Read message from the console and send it to the client.

Client:

1. Include the necessary header files.
2. Create a socket using socket function with family AF_INET, type as SOCK_STREAM.
3. Initialize server address to 0 using the bzero function.
4. Assign the sin_family to AF_INET.
5. Get the server IP address and the Port number from the console.
6. Using gethostbyname function assign it to a hostent structure, and assign it to sin_addr of the server address structure.
7. Request a connection from the server using the connect function.
8. Fork the process to receive message from the server and print it on the console.
9. Read message from the console and send it to the server.

Sample Output

Server:
(Host Name:Root1)

```
[root@localhost 4ita33]# vi fserver.c
[root@localhost 4ita33]# cc fserver.c
[root@localhost 4ita33]# ./a.out
Server is running.......
Enter the input data:
Received message from the client:hi
how are u
Data sent...
Enter the input data:
Received message from the client:i am fine
```

Client:
(Host Name:Root2)

```
[root@localhost 4ita33]# vi fclient.c
[root@localhost 4ita33]# cc fclient.c
[root@localhost 4ita33]# ./a.out 127.0.0.1
Enter the input data:
hi
Data sent...
Enter the input data:
Received message from the server:how are u
i am fine
Data sent...
Enter the input data:
```

INFERENCE:

Thus the chat application full duplex communication is established by sending the request from the client to the server, server gets the message and gives response to the client and prints it.

Program 4

SIMULATE TWO PHASE COMMIT PROTOCOL

Problem Definition

Two phase commit protocol is used for making a concensus when transactions are executed in distributed environments.

Problem Description

It is a distributed algorithm that coordinates all the processes that participate in a distributed atomic transaction on whether to commit or abort (roll back) the transaction.

The protocol works in the following manner: one node is designated the coordinator, which is the master site, and the rest of the nodes in the network are designated the cohorts. The protocol assumes that there is stable storage at each node with a write-ahead log, that no node crashes forever, that the data in the write-ahead log is never lost or corrupted in a crash, and that any two nodes can communicate with each other. The last assumption is not too restrictive, as network communication can typically be rerouted. The first two assumptions are much stronger; if a node is totally destroyed then data can be lost.

The protocol is initiated by the coordinator after the last step of the transaction has been reached. The cohorts then respond with an agreement message or an abort message depending on whether the transaction has been processed successfully at the cohort.

Algorithm

Commit request phase (or) voting phase:

1. The coordinator sends a query to commit message to all cohorts and waits until it has received a reply from all cohorts.
2. The cohorts execute the transaction up to the point where they will be asked to commit. They each write an entry to their undo log and an entry to their redo log.
3. Each cohort replies with an agreement message (cohort votes Yes to commit), if the transaction succeeded, or an abort message (cohort votes No, not to commit), if the transaction failed.

Commit phase (or) Completion phase:

Success

1. If the coordinator received an agreement message from all cohorts during the commit-request phase.
2. The coordinator sends a commit message to all the cohorts.
3. Each cohort completes the operation, and releases all the locks and resources held during the transaction.
4. Each cohort sends an acknowledgment to the coordinator.
5. The coordinator completes the transaction when acknowledgments have been received.

Failure

1. If any cohort sent an abort message during the commit-request phase:
2. The coordinator sends a rollback message to all the cohorts.
3. Each cohort undoes the transaction using the undo log, and releases the resources and locks held during the transaction.
4. Each cohort sends an acknowledgment to the coordinator.
5. The coordinator undoes the transaction when all acknowledgements have been received.

Expected Output:

```
C:\Documents and Settings\student\Desktop\DSLab\9>java Participant 7000
Recieved 'Can Commit?' from coordinator.
Random number generated is 0
Sent 'Yes Commit' to coordinator.
Recieved 'Commit' from coordinator.
Participant committed.

C:\Documents and Settings\student\Desktop\DSLab\9>_
```

```
C:\Documents and Settings\student\Desktop\DSLab\9>java Participant 7001
Recieved 'Can Commit?' from coordinator.
Random number generated is 4
Sent 'Yes Commit' to coordinator.
Recieved 'Commit' from coordinator.
Participant committed.

C:\Documents and Settings\student\Desktop\DSLab\9>
```

```
C:\Documents and Settings\student\Desktop\DSLab\9>java Participant 7002
Recieved 'Can Commit?' from coordinator.
Random number generated is 6
Sent 'Yes Commit' to coordinator.
Recieved 'Commit' from coordinator.
Participant committed.

C:\Documents and Settings\student\Desktop\DSLab\9>_
```

```
C:\Documents and Settings\student\Desktop\DSLab\9>java Coordinator 3
Asking the participant whether to commit.
1
Asking the participant whether to commit.
2
Asking the participant whether to commit.
3
Recieved 'Yes Commit' from participant.
Recieved 'Yes Commit' from participant.
Asking participant to Commit.
Asking participant to Commit.
coordinator rcvs Acknolodgement
Coordinator committed.
Recieved 'Yes Commit' from participant.
Asking participant to Commit.
coordinator rcvs Acknolodgement
coordinator rcvs Acknolodgement
Coordinator committed.
Coordinator committed.

C:\Documents and Settings\student\Desktop\DSLab\9>_
```

Program 5

DNS PROTOCOL USING RMI

Problem Definition

A server has to be designed that takes a string in input and returns another string in output depending upon the given input string.

Problem Description

A 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.

Here in this application the client sends the string which is the name of the website and server returns the IP address of that website.

Java RMI - Remote Method Invocation

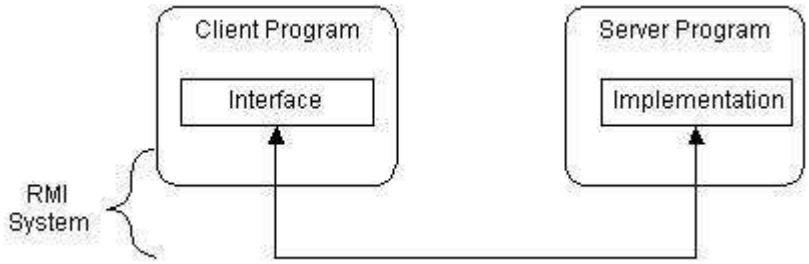
- A primary goal of RMI is to allow programmers to develop distributed Java programs
- RMI is the Java Distributed Object Model for facilitating communications among distributed objects
- RMI is a higher-level API built on top of sockets
- Socket-level programming allows you to pass data through sockets among computers
- RMI enables you not only to pass data among objects on different systems, but also to invoke methods in a remote object

The Differences between RMI and Traditional Client/Server Approach

- RMI component can act as both a client and a server, depending on the scenario in question.
- RMI system can pass functionality from a server to a client and vice versa. A client/server system typically only passes data back and fourth between server and client.

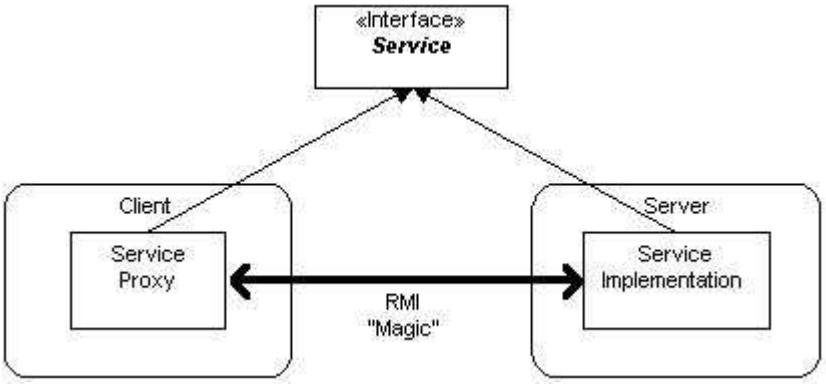
Interfaces: The Heart of RMI

- In RMI, the definition of a remote service is coded using a Java interface.
- The implementation of the remote service is coded in a class
- The key to understanding RMI is to remember that *interfaces define behavior* and *classes define implementation.*



RMI supports two classes:

1. The first class is the implementation of the behavior, and it runs on the server.
2. The second class acts as a proxy for the remote service and it runs on the client.



How it works A client program makes method calls on the proxy object, RMI sends the request to the remote JVM, and forwards it to the implementation.

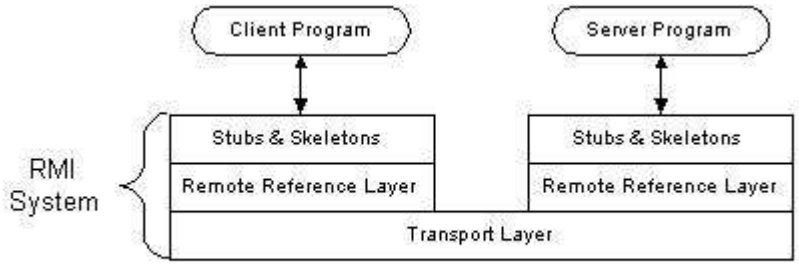
Any return values provided by the implementation are sent back to the proxy and then to the client's program.

RMI Architecture Layers

The RMI implementation is built from three abstraction layers.
The first is the **Stub and Skeleton layer**. This layer intercepts method calls made by the client to the interface reference variable and redirects these calls to a remote RMI service.

The next layer is the **Remote Reference Layer**. This layer understands how to interpret and manage references made from clients to the remote service objects. In JDK 1.1, this layer connects clients to remote service objects that are running and exported on a server. The connection is a one-to-one (unicast) link. In the Java 2 SDK, this layer was enhanced to support the activation of dormant remote service objects via *Remote Object Activation*.

The **Transport Layer** is based on TCP/IP connections between machines in a network. It provides basic connectivity, as well as some firewall penetration strategies.



RMI works as follows:

1. A server object is registered with the RMI registry.
2. A client looks through the RMI registry for the remote object.
3. Once the remote object is located, its stub is returned in the client.
4. The remote object can be used as a local object. The communication between client and server happens in terms of stubs and skeleton

RMI system is already designed, you take the following steps to build a system:

1. Write and compile Java code for interfaces
2. Write and compile Java code for implementation classes
3. Generate Stub and Skeleton class files from the implementation classes
4. Write Java code for a remote service host program
5. Develop Java code for RMI client program
6. Install and run RMI system

Pseudocode for RMI: The interface is specified using an RMI remote interface for DNS services:

```
public interface DNSInterface extends Remote {

    // update()
    //
    // The parameters specify a new entry in the SDNS directory
    //
    public void update(String ipAddress, string domainName) throws RemoteException;

    //request()
    //
    // The method returns the ip address corresponding to specified domain name
    //
    public string request(string domainName) throws RemoteException;

}
```

Implementation of SDNS Protocol You are to implement a client program and a server program that realize the SDNS protocol. The protocol is to be implemented using Java RMI. It may be helpful to refer to the RMI tutorial from the last lab.

SDNS Server The internal table that holds the (IP address, Domain Name) pairs can be implemented very straightforwardly using a Hashmap object. The Domain names can be used as keys to look up the stored IP addresses. The server should be registered in the RMI Registry using the name "sdns".

SDNS Client The SDNS client program can be invoked with either one or two arguments in the following format:
sdnslookup *sdns_server ip_address domain_name*
sdnslookup *sdns_server domain_name*

sdns_server this argument is the IP address of the SDNS server in decimal dot notation.

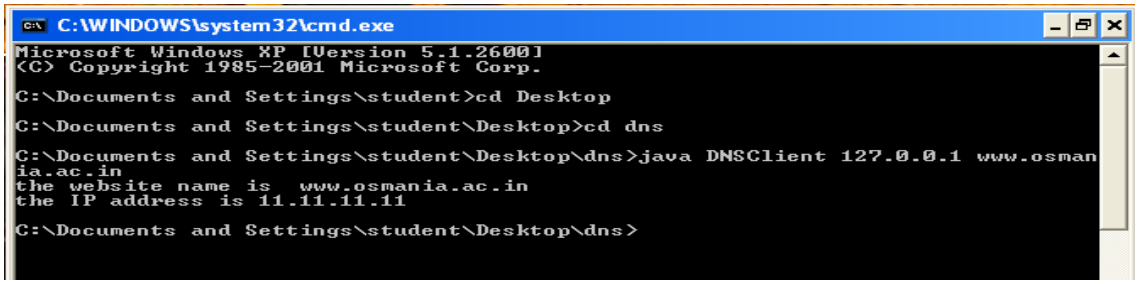
ip_address this argument is an IP address in decimal dot notation, e.g. 137.94.10.30.

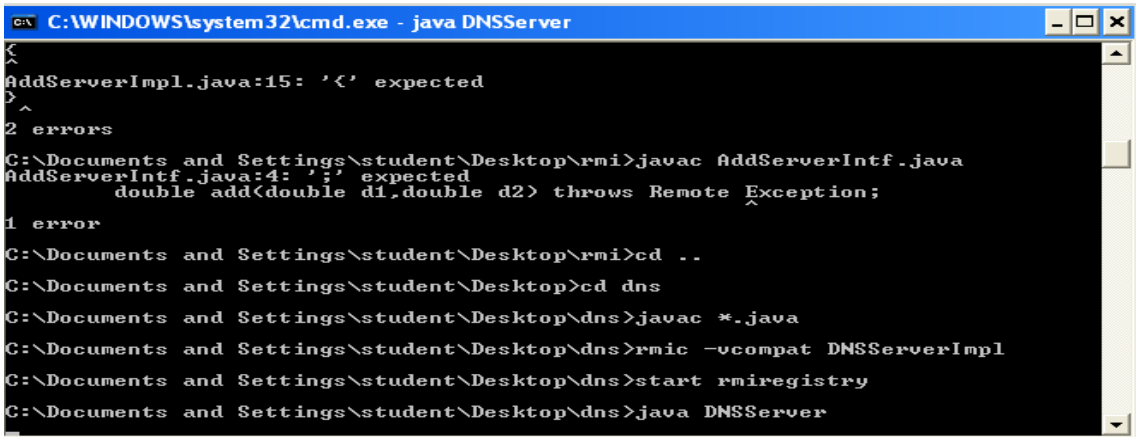
domain_name this argument is a fully qualified domain name, e.g. www.rmc.ca.

Depending on how many arguments the client program is called with it will either attempt to update the table in the SDNS server with a new (IP address, Domain Name) pair, or to request the IP address for a given domain name.

Once you have your client and server working you should try using your programs in combination with the SDNS programs of the other students. Provided you have implemented in accordance with the specified SDNS protocol your programs should work together.

OUTPUT


C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\student>cd Desktop
C:\Documents and Settings\student\Desktop>cd dns
C:\Documents and Settings\student\Desktop\dns>java DNSClient 127.0.0.1 www.osmania.ac.in
the website name is www.osmania.ac.in
the IP address is 11.11.11.11
C:\Documents and Settings\student\Desktop\dns>


C:\WINDOWS\system32\cmd.exe - java DNSServer
{
^
AddServerImpl.java:15: '{' expected
}
^
2 errors
C:\Documents and Settings\student\Desktop\rmi>javac AddServerIntf.java
AddServerIntf.java:4: ';' expected
double add(double d1,double d2) throws Remote Exception;
^
1 error
C:\Documents and Settings\student\Desktop\rmi>cd ..
C:\Documents and Settings\student\Desktop>cd dns
C:\Documents and Settings\student\Desktop\dns>javac *.java
C:\Documents and Settings\student\Desktop\dns>rmic -vcompat DNSServerImpl
C:\Documents and Settings\student\Desktop\dns>start rmiregistry
C:\Documents and Settings\student\Desktop\dns>java DNSServer