Course Syllabus

1	Simulate the functioning of Lamport's Logical Clock in 'C'.
2	Simulate the Distributed Mutual Exclusion in 'C'.
3	Implement a Distributed Chat Server using TCP Sockets in 'C'.
4	Implement RPC mechanism for a file transfer across a network in 'C'
5	Implement 'Java RMI' mechanism for accessing methods of remote systems.
6	Simulate Balanced Sliding Window Protocol in 'C'.
7	Implement CORBA mechanism by using 'C++' program at one end and 'Java program on the other.

Content Beyond Syllabus:

1.	Develop a client server application which implements File Transfer protocol.
2.	Implement a client server application which implements Name Server.

Experiment No.:1

Objective: Simulate the functioning of Lamport's Logical Clock in 'C'.

Theory: Lamport's Logical Clock is a mechanism for ordering events in a distributed system. Each process in the system maintains a logical clock that is incremented with each event, and it is used to establish a partial ordering of events. The logical clock values are used to determine the order of events, even if they occur on different processes.

Code:

```
#include <stdio.h>

// Structure to represent a process

typedef struct {

   int id;

   int logicalClock;

} Process;

// Function to simulate an event in a process

void simulateEvent(Process *process) {

   process->logicalClock++;

}

// Function to simulate message passing between two processes

void sendMessage(Process *sender, Process *receiver) {

   receiver->logicalClock = sender->logicalClock > receiver->logicalClock ? sender->logicalClock + 1 : receiver->logicalClock + 1;

}

// Function to print the logical clock of a process

void printLogicalClock(Process *process) {

   printf("Process %d - Logical Clock: %d\n", process->id, process->logicalClock);

}

int main() {

   // Create two processes
```

```
    Process process1 = {1, 0};
    Process process2 = {2, 0};
    // Simulate events and message passing
    simulateEvent(&process1);
    sendMessage(&process1, &process2);
    simulateEvent(&process1);
    simulateEvent(&process2);
    sendMessage(&process2, &process1);
    simulateEvent(&process2);
    // Print logical clocks of both processes
    printLogicalClock(&process1);
    printLogicalClock(&process2);
    return 0;
}
```

Output:

```
Process 1 - Logical Clock: 4
Process 2 - Logical Clock: 4

...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment No.:2

Objective: Simulate the Distributed Mutual Exclusion in 'C'

Theory: Distributed Mutual Exclusion is a challenging problem in distributed systems where multiple processes compete for access to a shared resource. One of the classical algorithms to achieve distributed mutual exclusion is the Ricart-Agrawala algorithm. Below is a simple simulation of the Ricart-Agrawala algorithm in C. Note that this is a basic example and might need modifications for a real distributed environment.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <pthread.h>
#define N 5 // Number of processes
typedef struct {
    bool requesting;
    bool in_cs;
    int timestamp;
} Process;
Process processes[N];
int clock = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t condition = PTHREAD_COND_INITIALIZER;
void request_critical_section(int process_id) {
    pthread_mutex_lock(&mutex);
    processes[process_id].requesting = true;
    processes[process_id].timestamp = clock;
    // Broadcast the request to all other processes
    for (int i = 0; i < N; ++i) {
        if (i != process_id) {
            // Send a request message
            // In a real distributed environment, this would involve network communication
        }
    }
```

```
    // Wait for replies from all other processes
    while (true) {
        int replies = 0;
        for (int i = 0; i < N; ++i) {
            if (i != process_id && processes[i].in_cs) {
                continue;
            }
            if (i != process_id && processes[i].requesting && processes[i].timestamp < processes[process_id].timestamp) {
                // Receive a reply
                replies++;
            }
        }
        if (replies == N - 1) {
            break;
        }
        pthread_cond_wait(&condition, &mutex);
    }
    pthread_mutex_unlock(&mutex);
}
void release_critical_section(int process_id) {
    pthread_mutex_lock(&mutex);
    processes[process_id].requesting = false;
    processes[process_id].in_cs = false;
    // Broadcast the release to all other processes
    for (int i = 0; i < N; ++i) {
        if (i != process_id) {
            // Send a release message
            // In a real distributed environment, this would involve network communication
        }
    }
    pthread_cond_broadcast(&condition);
    pthread_mutex_unlock(&mutex);

    }
```

```
void* process_function(void* arg) {
    int process_id = *((int*)arg);
    while (true) {
        sleep(rand() % 5 + 1); // Simulate some processing time
        request_critical_section(process_id);
        // Critical Section
        printf("Process %d enters the critical section.\n", process_id);
        processes[process_id].in_cs = true;
        sleep(rand() % 3 + 1); // Simulate some work in the critical section
        processes[process_id].in_cs = false;
        printf("Process %d exits the critical section.\n", process_id);
        release_critical_section(process_id);
        // Non-Critical Section
        sleep(rand() % 5 + 1);  // Simulate some processing time outside the critical section
    }
    return NULL;
}
int main() {
    pthread_t threads[N];
    int process_ids[N];
    srand(time(NULL));
    for (int i = 0; i < N; ++i) {
        process_ids[i] = i;
        pthread_create(&threads[i], NULL, process_function, (void*)&process_ids[i]);
    }
    for (int i = 0; i < N; ++i) {
        pthread_join(threads[i], NULL);
    }
    return 0;
}
```

Output:

```
Process 0 enters the critical section.
Process 0 exits the critical section.
Process 1 enters the critical section.
Process 1 exits the critical section.
Process 2 enters the critical section.
Process 2 exits the critical section.
...
```

Experiment No.:3

Objective: Implement a Distributed Chat Server using TCP Sockets in 'C'.

Theory: Creating a distributed chat server involves handling connections from multiple clients and facilitating communication between them. Below is a simple example of a distributed chat server in C using TCP sockets.

Code:

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
// Function designed for chat between client and server.
void func(int sockfd) {
char buff[MAX];
int n;
// infinite loop for chat
for (;;) {
bzero(buff, MAX);
// read the message from client and copy it in buffer
read(sockfd, buff, sizeof(buff));
// print buffer which contains the client contents
printf("From client: %s\t To client : ", buff);
bzero(buff, MAX);
n = 0;
// copy server message in the buffer
while ((buff[n++] = getchar()) != '\n');
// and send that buffer to client
write(sockfd, buff, sizeof(buff));
```

```
// if msg contains "Exit" then server exit and chat ended.
if (strncmp("exit", buff, 4) == 0) {
printf("Server Exit...\n");
break; } }
}
// Driver function
int main() {
int sockfd, connfd, len;
struct sockaddr_in servaddr, cli;
// socket create and verification
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd == -1) {
printf("socket creation failed...\n");
exit(0); }
else
printf("Socket successfully created..\n");
bzero(&servaddr, sizeof(servaddr));
// assign IP, PORT
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);
// Binding newly created socket to given IP and verification
if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
printf("socket bind failed...\n");
exit(0); }
else
printf("Socket successfully binded..\n");
// Now server is ready to listen and verification
if ((listen(sockfd, 5)) != 0) {
printf("Listen failed...\n");
exit(0); }
else
printf("Server listening..\n");
```

```
len = sizeof(cli);
// Accept the data packet from client and verification
connfd = accept(sockfd, (SA*)&cli, &len);
if (connfd < 0) {
printf("server accept failed...\n");
exit(0); }
else
printf("server accept the client...\n");
// Function for chatting between client and server
func(connfd);
// After chatting close the socket
close(sockfd); }
```

Client side:

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd) {
char buff[MAX];
int  n;
for (;;) {
bzero(buff, sizeof(buff));
printf("Enter the string : ");
n = 0;
while ((buff[n++] = getchar()) != '\n');
write(sockfd, buff, sizeof(buff));
bzero(buff,  sizeof(buff));
read(sockfd, buff, sizeof(buff));
printf("From Server : %s", buff);
```

```
if ((strncmp(buff, "exit", 4)) == 0) {
printf("Client Exit...\n");
break; } }
}
int main() {
int sockfd, connfd;
struct sockaddr_in servaddr, cli;
// socket create and verification
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd == -1) {
printf("socket creation failed...\n");
exit(0); }
else
printf("Socket successfully created..\n");
bzero(&servaddr, sizeof(servaddr));
// assign IP, PORT
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
servaddr.sin_port = htons(PORT);
// connect the client socket to server socket
if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
printf("connection with the server failed...\n");
exit(0); }
else
printf("connected to the server..\n");
// function for chat
func(sockfd);
// close the socket
close(sockfd);
}
```

Output:

Output

```
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
From client: hi
     To client : hello
From client: exit
     To client : exit
Server Exit...
```

Output

```
Socket successfully created..
connected to the server..
Enter the string : hi
From Server : hello
Enter the string : exit
From Server : exit
Client Exit...
```

Experiment No.:4

Objective: Implement RPC mechanism for a file transfer across a network in 'C'

Theory: Implementing a Remote Procedure Call (RPC) mechanism for file transfer in C involves creating a client-server architecture where the client requests the server to perform file transfer operations on its behalf.

Code:

CLIENT SIDE

```
#include "transfer.h"
#include <time.h>
void transfer_1(char *host, char *filetotransf) {
CLIENT *clnt;
int *result_1;
file transf_1_arg;
FILE *ofile;
long long int total = 0;
clnt = clnt_create (host, TRANSFER, TRANSFER_1, "tcp");
if (clnt == NULL) {
clnt_pcreateerror (host);
exit (1); }
ofile = fopen(filetotransf, "rb");
if(ofile == NULL) {
printf("File not found.\n");
exit(1); }
printf("Sending file %s.\n", filetotransf);
strcpy(transf_1_arg.name, filetotransf);
clock_t begin = clock();
while(1) {
transf_1_arg.nbytes = fread(transf_1_arg.data, 1, MAXLEN, ofile);
total += transf_1_arg.nbytes;
//printf("\r%lld bytes of %s sent to server.", total, transf_1_arg.name);
result_1 = transf_1(&transf_1_arg, clnt);
if (result_1 == (int *) NULL) {
clnt_perror (clnt, "call failed");
```

```
}
if(transf_1_arg.nbytes < MAXLEN) {
printf("\nUpload finished.\n");
break; } }
clock_t end = clock();
double upload_time = (double)(end - begin) / CLOCKS_PER_SEC;
printf("Upload time: %lf\n", upload_time);
clnt_destroy (clnt);
fclose(ofile); }
int main (int argc, char *argv[]) {
char *host;
char *filetotransf;
if (argc < 3) {
printf ("usage: %s <server_host> <file>\n", argv[0]);
exit (1); }
host = argv[1];
filetotransf = argv[2];
transfer_1 (host, filetotransf);
exit (0);
}
```

SERVER SIDE

```
#include "transfer.h"
char opened_file[MAXLEN];
FILE *ofile;
long long int total = 0;
int *
transf_1_svc(file *argp, struct svc_req *rqstp) {
static int result;
static char tempName[MAXLEN];
/*
* insert server code here
*/
/*
```

```
strcpy(tempName, "uploaded_");
strcat(tempName, argp->name);
strcpy(argp->name, tempName);
*/
total += argp->nbytes;
if (strcmp(opened_file, "") == 0 && ofile == NULL) {
printf("Receiving new file %s.\n", argp->name);
strcpy(opened_file, argp->name);
ofile = fopen(argp->name, "ab+");
}
if (strcmp(opened_file, argp->name) == 0) {
//printf("\r%lld bytes of file %s were received.", total, argp->name);
fflush(stdout);
fwrite(argp->data, 1, argp->nbytes, ofile);
if (argp->nbytes < MAXLEN) {
printf("\nFinished receiving %s.\n", argp->name);
total = 0;
fclose(ofile);
ofile = NULL;
strcpy(opened_file, ""); }
}
return &result;
}
```

Output:

Server Side

```
Socket file descriptor 3 received
Successfully binded!
Waiting for file name...
File Name Received: dm.txt
File Successfully opened!
Waiting for file name...
File Name Received: /home/dmayank/Documents/dm.txt
File Successfully opened!
```

Client Side

```
Socket file descriptor 3 received

Please enter file name to receive:
dm.txt

---------Data Received---------
30
--------------------------------

Please enter file name to receive:
/home/dmayank/Documents/dm.txt

---------Data Received---------
30
--------------------------------
```

Experiment No.:5

Objective: Implement 'Java RMI' mechanism for accessing methods of remote systems.

Theory: Java RMI (Remote Method Invocation) allows you to invoke methods on objects that reside in another address space, often on a different machine. Below is a simple example of a Java RMI server and client. In this example, we'll create a remote interface, a server implementation, and a client that invokes methods on the remote server.

Code:

Create the remote interface (Calculator.java):

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Calculator extends Remote {
    int add(int a, int b) throws RemoteException;
    int subtract(int a, int b) throws RemoteException;
}
```

Create the server implementation (CalculatorImpl.java):

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class CalculatorImpl extends UnicastRemoteObject implements Calculator {
    public CalculatorImpl() throws RemoteException {
        super(); }
    @Override
    public int add(int a, int b) throws RemoteException {
        return a + b;    }
    @Override
    public int subtract(int a, int b) throws RemoteException {
        return a - b;  }
}
```

Create the server application (Server.java):

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
public class Server {
    public static void main(String[] args) {
        try {
```

```
        // Create and export the remote object
        Calculator calculator = new CalculatorImpl();
        Registry registry = LocateRegistry.createRegistry(1099);
        registry.rebind("CalculatorService", calculator);
        System.out.println("Server is ready.");
    } catch (Exception e) {
        System.err.println("Server exception: " + e.toString());
        e.printStackTrace();    }  }
}
```

Create the client application (Client.java):

```
import java.rmi.registry.LocateRegistry;
import  java.rmi.registry.Registry;
public class Client {
  public static void main(String[] args) {
    try {
      // Get the remote object reference
      Registry registry = LocateRegistry.getRegistry("localhost", 1099);
      Calculator calculator = (Calculator) registry.lookup("CalculatorService");
      // Invoke remote methods
      int resultAdd = calculator.add(10, 5);
      int resultSubtract = calculator.subtract(10, 5);
      System.out.println("Result of addition: " + resultAdd);
      System.out.println("Result of subtraction: " + resultSubtract);
    } catch (Exception e) {
      System.err.println("Client exception: " + e.toString());
      e.printStackTrace();    }  }
}
```

Output:

```
Server is ready.
Result of addition: 15
Result of subtraction: 5
```

Experiment No.:6

Objective: Simulate Balanced Sliding Window Protocol in 'C'.

Theory: The Balanced Sliding Window Protocol is a protocol used for reliable communication over a network. It ensures that data sent by the sender is received correctly by the receiver, and it handles issues such as packet loss and out-of-order delivery.

Code:

```
#include  <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define WINDOW_SIZE 4
typedef struct {
   int frame;
   bool ack;
} Frame;
void sender(Frame frames[], int totalFrames) {
   int base = 0;
   int nextSeqNum = 0;
   while (base < totalFrames) {
      // Send frames within the window
      for (int i = base; i < base + WINDOW_SIZE && i < totalFrames; i++) {
         frames[i].frame = nextSeqNum;
         frames[i].ack = false;
         printf("Sender: Sending Frame %d\n", frames[i].frame);
         nextSeqNum++;
      }
      // Simulate transmission over a network
      // In a real-world scenario, this would be replaced with actual network communication
      // Simulate acknowledgment reception
      for (int i = base; i < base + WINDOW_SIZE && i < totalFrames; i++) {
         frames[i].ack = true;
         printf("Sender: Received ACK for Frame %d\n", frames[i].frame);
      }
      // Move the window based on received acknowledgments
```

```
        while (base < totalFrames && frames[base].ack) {
            base++;
        }
    }
    printf("Sender: All frames sent and acknowledged.\n");
}
void receiver(Frame frames[], int totalFrames) {
    int expectedSeqNum = 0;
    for (int i = 0; i < totalFrames; i++) {
        if (frames[i].frame == expectedSeqNum) {
            printf("Receiver: Received Frame %d\n", frames[i].frame);
            expectedSeqNum++;
        } else {
            printf("Receiver: Received Out-of-Order Frame %d, Discarding\n", frames[i].frame);
        }
    }
    printf("Receiver: All frames received.\n");
}
int main() {
    int totalFrames = 12;
    Frame frames[totalFrames];
    printf("Simulation of Balanced Sliding Window Protocol\n\n");
    printf("Initializing frames:\n");
    for (int i = 0; i < totalFrames; i++) {
        frames[i].frame = -1;
        frames[i].ack = false;
    }
    printf("\nSender and Receiver Communication:\n\n");
    sender(frames, totalFrames);
    printf("\n");
    receiver(frames, totalFrames);
    return 0;
}
```

Output:

```
Simulation of Balanced Sliding Window Protocol

Initializing frames:

Sender and Receiver Communication:

Sender: Sending Frame 0
Sender: Sending Frame 1
Sender: Sending Frame 2
Sender: Sending Frame 3
Sender: Received ACK for Frame 0
Sender: Received ACK for Frame 1
Sender: Received ACK for Frame 2
Sender: Received ACK for Frame 3
Sender: Sending Frame 4
Sender: Sending Frame 5
Sender: Sending Frame 6
Sender: Sending Frame 7
Sender: Received ACK for Frame 4
Sender: Received ACK for Frame 5
Sender: Received ACK for Frame 6
Sender: Received ACK for Frame 7
Sender: Sending Frame 8
Sender: Sending Frame 9
Sender: Sending Frame 10
Sender: Sending Frame 11
Sender: Received ACK for Frame 8
Sender: Received ACK for Frame 9
```

```
Sender: Sending Frame 8
Sender: Sending Frame 9
Sender: Sending Frame 10
Sender: Sending Frame 11
Sender: Received ACK for Frame 8
Sender: Received ACK for Frame 9
Sender: Received ACK for Frame 10
Sender: Received ACK for Frame 11
Sender: All frames sent and acknowledged.

Receiver: Received Frame 0
Receiver: Received Frame 1
Receiver: Received Frame 2
Receiver: Received Frame 3
Receiver: Received Frame 4
Receiver: Received Frame 5
Receiver: Received Frame 6
Receiver: Received Frame 7
Receiver: Received Frame 8
Receiver: Received Frame 9
Receiver: Received Frame 10
Receiver: Received Frame 11
Receiver: All frames received.


...Program finished with exit code 0
Press ENTER to exit console.
Press ENTER to exit console.
```

Experiment No.:7

Objective: Implement CORBA mechanism by using 'C++' program at one end and 'Java program on the other

Theory: Creating a CORBA-based client-server application involves several steps, including defining an Interface Definition Language (IDL) file, generating stubs and skeletons, implementing the server in C++, and creating the client in Java.

Code:

1. IDL File (example.idl):

```
module Example {
   interface Calculator {
      long add(in long a, in long b);
      long subtract(in long a, in long b);
   };
};
```

2. Generate Stubs and Skeletons:

```
# Generate C++ skeletons
idlc -S example.idl
# Generate Java stubs
idlj example.idl
```

3. C++ Server (CalculatorServer.cpp):

```
#include <iostream>
#include "Example.hh"
class CalculatorImpl : public POA_Example::Calculator {
public:
   virtual CORBA::Long add(CORBA::Long a, CORBA::Long b) { return a + b;    }
   virtual CORBA::Long subtract(CORBA::Long a, CORBA::Long b) {  return a - b;  }
};
int main(int argc, char* argv[]) {
   try {
      CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
      // Create the servant
```

```
        CalculatorImpl calculatorImpl;
        // Activate the servant
        PortableServer::POA_var poa = PortableServer::POA::_narrow(orb->resolve_initial_references("RootPOA"));
        PortableServer::ObjectId_var oid = poa->activate_object(&calculatorImpl);
        // Obtain a reference to the object
        CORBA::Object_var obj = poa->id_to_reference(oid);
        // Register the object reference with the naming service (if needed) and Run the ORB
        orb->run();
    } catch (const CORBA::Exception& e) {
        std::cerr << "Exception: " << e << std::endl;
        return 1;    }
    return 0;
}
```

4. Java Client (CalculatorClient.java):

```
import Example.*;
public class CalculatorClient {
    public static void main(String[] args) {
        try {
            // Initialize the ORB
            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
            // Obtain the object reference from the naming service (if needed)
            // org.omg.CORBA.Object obj = orb.string_to_object("IOR:PUT_YOUR_IOR_STRING_HERE");
            // Obtain the object reference directly (if not using naming service)
            org.omg.CORBA.Object obj = orb.string_to_object("corbaname::localhost:1050#Calculator");
            // Narrow the object reference to the Calculator interface
            Calculator calculator = CalculatorHelper.narrow(obj);
            long resultAdd = calculator.add(10, 5);
            long resultSubtract = calculator.subtract(10, 5);
            System.out.println("Result of addition: " + resultAdd);
            System.out.println("Result of subtraction: " + resultSubtract);
        } catch (Exception e) {
            e.printStackTrace();        } }}
```

Experiment No.:1

Objective: Develop a client server application which implements File Transfer protocol.

Theory: (FTP) is a standard network protocol used to transfer computer files from one host to another host over a TCP-based network, such as the internet.

FTP is built on client-server architecture and used separate control and data connections between the client and the server. FTP users may authenticate themselves using a clear-text sing-in-protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. For secure transmission that protects the username and password and encrypts the content, FTP is often secured with SSL/TLS. SSH File Transfer Protocol is sometimes also used instead, but is technologically different.

Code:

FTP Client:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
class One extends JFrame implements ActionListener{
public JButton b,b1; public JLabel l;
public JLabel l1,lmsg1,lmsg2; One(){
b=new JButton("Upload");
l=new JLabel("Uplaod a file : ");
lmsg1=new JLabel("");
b1=new JButton("Download");
l1=new JLabel("Downlaod a file");
```

```
lmsg2=new JLabel("");
setLayout(new GridLayout(2,3,10,10));
add(l);add(b);add(lmsg1);add(l1);add(b1);add(lmsg2);
b.addActionListener(this);
b1.addActionListener(this);
setVisible(true); setSize(600,500);
}
public void actionPerformed(ActionEvent e) {
// TODO Auto-generated method stub try {
/* String s=e.getActionCommand();
if(s.equals("Upload"))*/
if (b.getModel().isArmed()) {
Socket s=new Socket("localhost",1010);
System.out.println("Client connected to server");
JFileChooser j=new JFileChooser();
int val;
val=j.showOpenDialog(One.this);
String filename=j.getSelectedFile().getName();
String path=j.getSelectedFile().getPath();
PrintStream out=new PrintStream(s.getOutputStream());
out.println("Upload");
out.println(filename);
FileInputStream fis=new FileInputStream(path); int n=fis.read();
while (n!=-1) {
        out.print((char)n);n=fis.read(); }
fis.close(); out.close();lmsg1.setText(filename+"is uploaded");
//s.close(); repaint();
}
if (b1.getModel().isArmed()) {
Socket s=new Socket("localhost",1010);
System.out.println("Client connected to server");
String remoteadd=s.getRemoteSocketAddress().toString();
System.out.println(remoteadd);
```

```
JFileChooser j1=new JFileChooser(remoteadd);
int val;
val=j1.showOpenDialog(One.this);
String filename=j1.getSelectedFile().getName();
String filepath=j1.getSelectedFile().getPath();
System.out.println("File name:"+filename);
PrintStream out=new PrintStream(s.getOutputStream());
out.println("Download");
out.println(filepath);
FileOutputStream fout=new FileOutputStream(filename);
DataInputStream fromserver=new DataInputStream(s.getInputStream());
int ch;
while ((ch=fromserver.read())!=-1) {
fout.write((char) ch); }
fout.close();
//s.close();
lmsg2.setText(filename+"is downlaoded");
repaint(); } }
catch (Exception ee) {
// TODO: handle exception System.out.println(ee); }
} }
public class FTPClient {
public static void main(String[] args) {
new One(); }
}
```

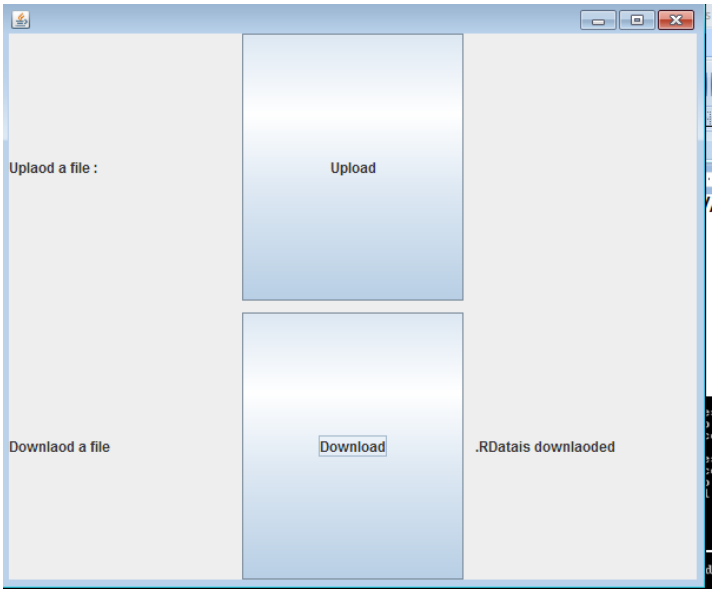
FTP Server:

```
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
```

```
public class FTPServer {
public static void main(String[] args) {
try {
while (true) {
ServerSocket ss=new ServerSocket(1010);
Socket  sl=ss.accept();
System.out.println("Server  scoket  is  created. ");
System.out.println(" test1");
DataInputStream fromserver=new DataInputStream(sl.getInputStream());
System.out.println(" test2");
String option=fromserver.readLine();
if (option.equalsIgnoreCase("upload")) {
System.out.println("upload test");
String filefromclient=fromserver.readLine();
File clientfile=new File(filefromclient);
FileOutputStream fout=new FileOutputStream(clientfile);
int ch;
while ((ch=fromserver.read())!=-1) {
fout.write((char)ch);
}
fout.close();
}
if (option.equalsIgnoreCase("download")){
System.out.println("download test");
String filefromclient=fromserver.readLine();
File clientfile=new File(filefromclient);
FileInputStream fis=new FileInputStream(clientfile);
PrintStream out=new PrintStream(sl.getOutputStream());
int n=fis.read();
while  (n!=-1){
out.print((char)n); n=fis.read(); }
fis.close();
out.close();
```

```
} //while

} }

catch (Exception e) {

System.out.println(e);

// TODO: handle exception

}}}
```

Output:




```
C:\Users\LAB4-57\Desktop>java FTPClient
Client connected to server
java.net.ConnectException: Connection refused: connect

C:\Users\LAB4-57\Desktop>java FTPClient
java.net.ConnectException: Connection refused: connect
Client connected to server
localhost/127.0.0.1:1010
File name:.RData
```

```
C:\Users\LAB4-57>cd desktop

C:\Users\LAB4-57\Desktop>javac FTPServer.java
Note: FTPServer.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\LAB4-57\Desktop>java FTPServer
Server scoket is created....
 test1
 test2
upload test
java.net.BindException: Address already in use: JVM_Bind

C:\Users\LAB4-57\Desktop>java FTPServer
Server scoket is created....
 test1
 test2
download test
java.net.BindException: Address already in use: JVM_Bind

C:\Users\LAB4-57\Desktop>java FTPServer
```

Experiment No.:2

Objective: Implement a client server application which implements Name Server.

Theory: Name server is a client / server network communication protocol. Name server clients send request to the server while name servers send response to the client. Client request contain a name which is converted into in IP address known as a forward name server lookups while requests containing an IP address which is converted into a name known as reverse name server lookups. Name server implements a distributed database to store the name of all the hosts available on the internet. If a client like a web browser sends a request containing a hostname, then a piece of software such as name server resolver sends a request to the name server to obtain the IP address of a hostname. If name server does not contain the IP address associated with a hostname then it forwards the request to another name server. It IP address has arrived at the resolver, which in turn completes the request over the internet protocol.

Code:

```
import java.net.*;
import java.io.*;
import java.util.*;
public class DNS {
public static void main(String[] args) {
int n;
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
Do {
System.out.println("\n Menu: \n 1. DNS 2. Reverse DNS 3. Exit \n");
System.out.println("\n Enter your choice");
n = Integer.parseInt(System.console().readLine());
if(n==1) {
try {
System.out.println("\n Enter Host Name ");
String hname=in.readLine();
InetAddress address;
address = InetAddress.getByName(hname);
System.out.println("Host Name: " + address.getHostName());
System.out.println("IP: " + address.getHostAddress());
}
catch(IOException ioe) {
ioe.printStackTrace(); }
```

```
}
if(n==2) {
try {

System.out.println("\n Enter IP address");
String ipstr = in.readLine();
InetAddress ia = InetAddress.getByName(ipstr);
System.out.println("IP: "+ipstr);
System.out.println("Host Name: " +ia.getHostName());
}
catch(IOException ioe) {
ioe.printStackTrace(); }
}
}while(!(n==3));
}}
```

Output: