ARMIET
ALAMURI RATNAMALA
INSTITUTE OF ENGINEERING AND TECHNOLOGY
(Run by Koti Vidya Charitable Trust)
A. S. Rao Nagar, Sapgaon, Tal. Shahapur, Dist. Thane
Pin.: 421 601 Tel.: 02527 - 212221/22 Tel. Fax: 022-40244310

तेजस्विनावधीतमस्तु

Alamuri Ratnamala
Institute of Engineering and Technology

IT DEPARTMENT

SEMESTER VI

DISTRIBUTED SYSTEMS

LABORATORY MANUAL

AS PER REVISED SYLLABUS
UNIVERSITY OF MUMBAI

LABORATORY MANUAL CONTENTS

This manual is intended for the Third year students of Information Technology in the subject of Distributed Systems . This manual typically contains practical/Lab Sessions related Middleware covering various aspects related the subject to enhanced understanding.

Although, as per the syllabus, It covers all the aspects of d.s. It introduce its readers to basic concepts of middleware, states of art middleware technology and middleware services like RMI,CORBA,DCOM and EJB.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Best of Luck for your Laboratory Sessions

Mr.Likhesh Kolhe
HOD IT Department

Mr.Mayank Mangal
Asst. Prof IT Department

Do's and Don'ts in Laboratory:

1.Make entry in the Log Book as soon as you enter the Laboratory.

2.All the students should sit according to their roll numbers starting from their left to right.

3.All the students are supposed to enter the terminal number in the log book.

4.Do not change the terminal on which you are working.

5.All the students are expected to get at least the algorithm of the program/concept to be implemented.

6.Strictly observe the instructions given by the teacher/Lab Instructor.

Instruction for STUDENTS

1. Submission related to whatever lab work has been completed should be done during the next lab session. The immediate arrangements for printouts related to submission on the day of practical assignments.

2. Students should be taught for taking the printouts under the observation of lab teacher.

3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

DS LABORATORY

ACADEMIC YEAR 2014-2015

DEPARTMENT OF INFORMATION TECHNOLOGY

SUB: DISTRIBUTED SYSTEMS

Class: Third Year **Sem: VI**

INDEX

PREPARED BY APPROVED BY

EXPERIMENT NO:1 DOP: DOS: GRADE:

TITLE OF EXPERIMENT: CLIENT-SERVER WITH RPC

AIM: TO STUDY CLIENT-SERVER BASED PROGRAM

AIM: To study Client Server based program using RPC.

THEORY:

RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional, or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports.

RPC makes the client/server model of computing more powerful and easier to program. When combined with the ONC RPCGEN protocol compiler clients transparently make remote calls through a local procedure interface.

An RPC is analogous to a function call. Like a function call, when an RPC is made, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure. Figure shows the flow of activity that takes place during an RPC call between two networked systems. The client makes a procedure call that sends a request to the server and waits. The thread is blocked from processing until either a reply is received, or it times out. When the request arrives, the server calls a dispatch routine that performs the requested service, and sends the reply to the client. After the RPC call is completed, the client program continues. RPC specifically supports network applications.

Program Code:

```
#include <stdio.h>
#include <rpc.h>
#include <pmapclnt.h>
#include <msg.h>

static void   messageprog_1();
static char   *printmessage_1();

static struct timeval TIMEOUT = { 25, 0 };

main()
{
   SVCXPRT *transp;

   (void)pmap_unset(MESSAGEPROG, MESSAGEVERS);

   transp = svcudp_create(RPC_ANYSOCK);
   if (transp == (SVCXPRT *)NULL)
   {
```

```
    (void)fprintf(stderr, "CANNOT CREATE UDP SERVICE.\n");
    exit(16);
  }
  if (!svc_register(transp, MESSAGEPROG, MESSAGEVERS, messageprog_1, IPPROTO_UDP))
  {
    (void)fprintf(stderr,
    "UNABLE TO REGISTER (MESSAGEPROG, MESSAGEVERS, UDP).\n");
    exit(16);
  }

  transp = svctcp_create(RPC_ANYSOCK, 0, 0);
  if (transp == (SVCXPRT *)NULL)
  {
    (void)fprintf(stderr, "CANNOT CREATE TCP SERVICE.\n");
    exit(16);
  }
  if (!svc_register(transp, MESSAGEPROG, MESSAGEVERS, messageprog_1, IPPROTO_TCP))
  {
    (void)fprintf(stderr,
    "UNABLE TO REGISTER (MESSAGEPROG, MESSAGEVERS, TCP).\n");
    exit(16);
  }
  svc_run();
  (void)fprintf(stderr, "SVC_RUN RETURNED\n");
  exit(16);
  return(0);
}

static void messageprog_1(rqstp, transp)
struct svc_req   *rqstp;
SVCXPRT  *transp;
{
  union
  {
    char  *printmessage_1_arg;
  }

  argument;
  char     *result;
  bool_t  (*xdr_argument)();
  bool_t  (*xdr_result)();
  char   *(*local)();

  switch (rqstp->rq_proc)
  {
    case NULLPROC:
      (void)svc_sendreply(transp, xdr_void, (char *)NULL);
      return;

    case PRINTMESSAGE:
      xdr_argument = xdr_wrapstring;
      xdr_result = xdr_int;
      local = (char *(*)()) printmessage_1;
      break;
```

```
      default:
         svcerr_noproc(transp);
         return;
   }
   bzero((char *)&argument, sizeof(argument));
   if (!svc_getargs(transp, xdr_argument, &argument))
   {
      svcerr_decode(transp);
      return;
   }
   result = (*local)(&argument, rqstp);
   if (result != (char *)NULL &&
      !svc_sendreply(transp, xdr_result, result))
   {
      svcerr_systemerr(transp);
   }
   if (!svc_freeargs(transp, xdr_argument, &argument))
   {
      (void)fprintf(stderr, "UNABLE TO FREE ARGUMENTS\n");
      exit(16);
   }
   return;
}

char  *printmessage_1(msg)
char  **msg;
{
   static char  result;

   fprintf(stderr, "%s\n", *msg);
   result = 1;
   return(&result);
}
```

Conclusion : Hence we have studied and run Client-Server based RPC program successfully.

EXPERIMENT NO:2 DOP: DOS: GRADE:

TITLE OF EXPERIMENT: CLIENT-SERVER WITH RMI

AIM: TO STUDY CLIENT-SERVER BASED PROGRAM

AIM: To study Client Server based program using RMI.

THEORY:

The RMI application comprises of the 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. The RMI application provides the mechanism by which the server and the client communicate and pass information back and forth. The RMI distributed application uses the RMI Registry to obtain a reference to a remote object. The server calls the registry to associate 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.

Program:

ReceiveMessageInterface.java

```
import java.rmi.*;

public interface ReceiveMessageInterface extends Remote{
  void receiveMessage(String x) throws RemoteException;
}
```

The above code defines the RMI interface. The **receiveMessage()** method is implemented in the server class.

Here is the code of RMI Server:

```
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import java.net.*;

public class RmiServer extends
  java.rmi.server.UnicastRemoteObject implements ReceiveMessageInterface{
  String address;
  Registry registry;

  public void receiveMessage(String x) throws RemoteException{
  System.out.println(x);
  }

  public RmiServer() throws RemoteException{
  try{
  address = (InetAddress.getLocalHost()).toString();
  }
  catch(Exception e){
  System.out.println("can't get inet address.");
  }
  int port=3232;
```

```
System.out.println("this address=" + address +  ",port=" + port);
try{
registry = LocateRegistry.createRegistry(port);
registry.rebind("rmiServer", this);
}
catch(RemoteException e){
System.out.println("remote exception"+ e);
}
}
static public void main(String args[]){
try{
RmiServer server = new RmiServer();
}
catch (Exception e){
e.printStackTrace();
System.exit(1);
}
}
}
```

The above class uses **LocateRegistry** class to create a remote object registry that accepts calls on a specific port

Output of the above program:

```
C:\rose>javac RmiServer.java
C:\rose>java RmiServer
this
address=roseindi/192.168.10.104,port=3232t=
_3232
```

Here is the code of RMI Client:

```
import java.rmi.*;
import java.rmi.registry.*;
import java.net.*;

public class RmiClient{
 static public void main(String args[]){
 ReceiveMessageInterface rmiServer;
 Registry registry;
 String serverAddress=args[0];
 String serverPort=args[1];
 String text=args[2];
 System.out.println
  ("sending " + text + " to " +serverAddress + ":" + serverPort);
 try{
 registry=LocateRegistry.getRegistry
 (serverAddress,(new Integer(serverPort)).intValue());
 rmiServer=(ReceiveMessageInterface)(registry.lookup("rmiServer"));
 // call the remote method
 rmiServer.receiveMessage(text);
 }
 catch(RemoteException e){
 e.printStackTrace();
 }
 catch(NotBoundException e){
 System.err.println(e);
```

```
  }
  }
}
```

lookup(): This is the method that returns a reference, a stub, for the remote object associated with the specified name.

Output of the above program:

```
C:\rose>java RmiClient 192.168.10.104
3232 roseindia
sending roseindia to 192.168.10.104:3232

C:\rose>
```

If the RMI client sends any type of massage then massage will be displayed on the RMI Server.

```
C:\rose>java RmiServer
this
address=roseindi/192.168.10.104,port=3232
roseindia
```

Conclusion : Hence we have studied and run Client-Server based RMI program successfully.

EXPERIMENT NO:3	**DOP:**	**DOS:**	**GRADE:**
TITLE OF EXPERIMENT: CLOCK SYNCHRONIZATION			
AIM:			

AIM: To Study Implementation of Clock Synchronization (logical/physical).

THEORY:

Introduction:

Steps:

EXPERIMENT NO:4	DOP:	DOS:	GRADE:
TITLE OF EXPERIMENT: ELECTION ALGO BY BULLY			
AIM: IMPLEMENTATION OF BULLY ELECTION ALGO			

AIM: To Study Implementation of Election algorithm.

THEORY:

The bully algorithm is a method in distributed computing for dynamically electing a coordinator by process ID number. The process with the highest process ID number is selected as the coordinator.

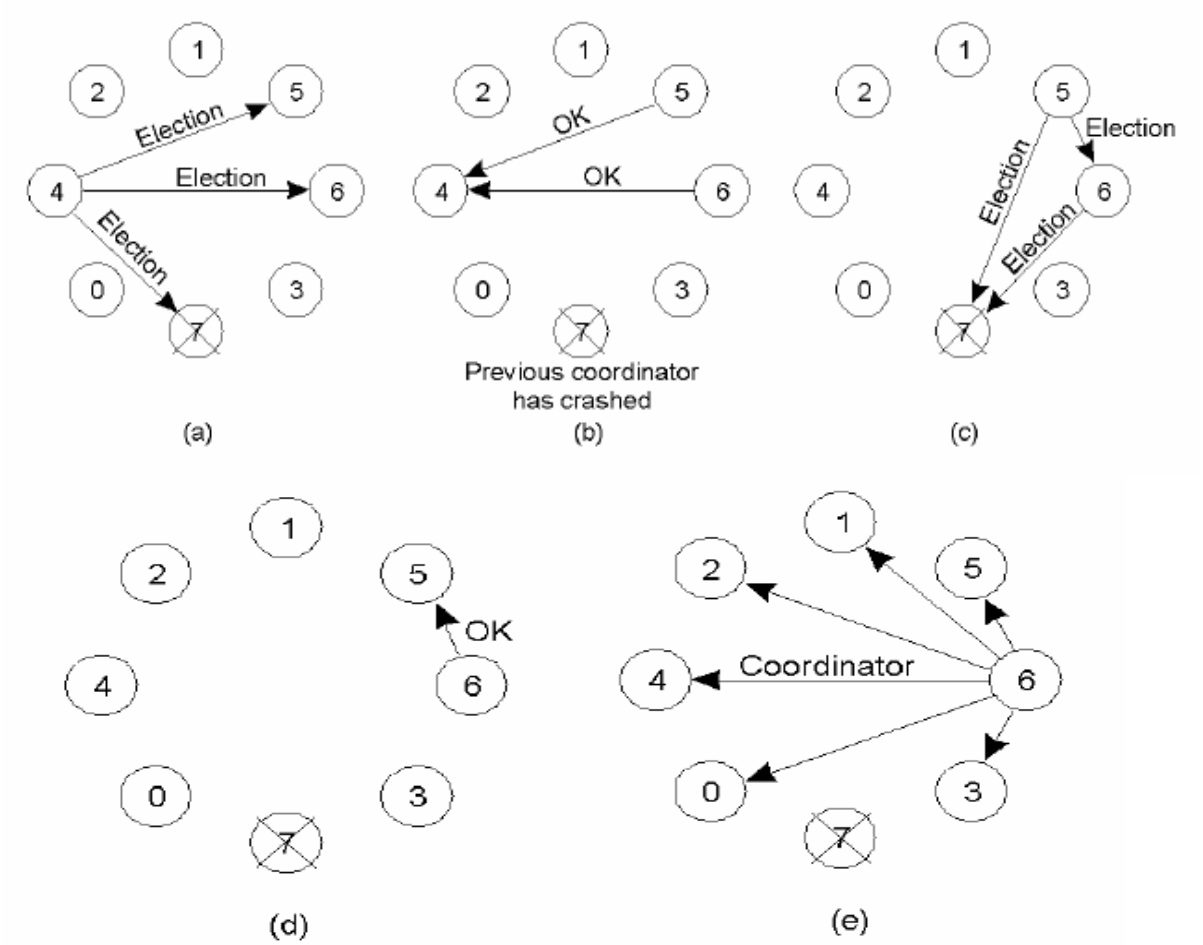
When a process P determines that the current coordinator is down because of message timeouts or failure of the coordinator to initiate a handshake, it performs the following sequence of actions:

- P broadcasts an election message (inquiry) to all other processes with higher process IDs, expecting an "I am alive" response from them if they are alive.
- If P hears from no process with a higher process ID than it, it wins the election and broadcasts victory.
- If P hears from a process with a higher ID, P waits a certain amount of time for any process with a higher ID to broadcast itself as the leader. If it does not receive this message in time, it re-broadcasts the election message.
- If P gets an election message (inquiry) from another process with a lower ID it sends an "I am alive" message back and starts new elections.
- Assumptions
- Each process knows the ID and address of every other process
- Communication is reliable
- A process initiates an election if it just recovered from failure or it notices that the coordinator has failed
- Three types of messages: *Election, OK, Coordinator*
- Several processes can initiate an election simultaneously
- Need consistent result

Details:

- Any process P can initiate an election
- P sends *Election* messages to all process with higher IDs and awaits *OK* messages
  - If no *OK* messages, P becomes coordinator and sends *Coordinator* messages to all processes with lower IDs
  - If it receives an *OK*, it drops out and waits for an *Coordinator* message
- If a process receives an *Election* message
  - Immediately sends *Coordinator* message if it is the process with highest ID
  - Otherwise, returns an *OK* and starts an election
- If a process receives a *Coordinator* message, it treats sender as the coordinator.

Example:



Program Code:

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>

struct process {
  int no;
  int priority;
  int active;
  struct process *next;
};
typedef struct process proc;

struct priority {
  int pri;
  struct priority *next;
```

```
  proc *pp;
};
typedef struct priority pri;

pri* find_priority(proc *head, pri *head1) {
  proc *p1;
  pri *p2, *p3;
  p1 = head;

  while (p1->next != head) {
    if (p1->active == 1) {
      if (head1 == NULL) {
        head1 = (pri*) malloc(sizeof(pri));
        head1->pri = p1->priority;
        head1->next = NULL;
        head1->pp = p1;
        p2 = head1;
      } else {
        p3 = (pri*) malloc(sizeof(pri));
        p3->pri = p1->priority;
        p3->pp = p1;
        p3->next = NULL;
        p2->next = p3;
        p2 = p2->next;
      }
      p1 = p1->next;
    } else
      p1 = p1->next;
  } //end while

  p3 = (pri*) malloc(sizeof(pri));
  p3->pri = p1->priority;
  p3->pp = p1;
  p3->next = NULL;
  p2->next = p3;
  p2 = p2->next;
  p3 = head1;

  return head1;
} //end find_priority()

int find_max_priority(pri *head) {
  pri *p1;
  int max = -1;
  int i = 0;
  p1 = head;

  while (p1 != NULL) {
    if (max < p1->pri && p1->pp->active == 1) {
      max = p1->pri;
      i = p1->pp->no;
    }
    p1 = p1->next;
  }
```

```
  return i;
}

void bully() {
  proc *head;
  proc *p1;
  proc *p2;
  int n, i, pr, maxpri, a, pid, max, o;
  char ch;

  head = p1 = p2 = NULL;

  printf("\nnEnter how many process: ");
  scanf("%d", &n);

  for (i = 0; i < n; i++) {
    printf("\nEnter priority of process %d: ", i + 1);
    scanf("%d", &pr);

    printf("\nIs process with id %d is active ?(0/1) :", i + 1);
    scanf("%d", &a);

  if (head == NULL) {
      head = (proc*) malloc(sizeof(proc));
      if (head == NULL) {
        printf("\nMemory cannot be allocated");
        getch();
        exit(0);
      }
      head->no = i + 1;
      head->priority = pr;
      head->active = a;
      head->next = head;
      p1 = head;
    } else {
      p2 = (proc*) malloc(sizeof(proc));
      if (p2 == NULL) {
        printf("\nMemory cannot be allocated");
        getch();
        exit(0);
      }
      p2->no = i + 1;
      p2->priority = pr;
      p2->active = a;
      p1->next = p2;
      p2->next = head;
      p1 = p2;
    }
  } //end for

  printf("\nEnter the process id that invokes election algorithm: ");
  scanf("%d", &pid);
  p2 = head;
  while (p2->next != head) {
```

```
  if (p2->no == pid) {
    p2 = p2->next;
    break;
  }
  p2 = p2->next;
}

printf("\nProcess with id %d has invoked election algorithm", pid);
printf("\t\nElection message is sent to processes");

while (p2->next != head) {
  if (p2->no > pid)
    printf("%d", p2->no);
  p2 = p2->next;
}

printf("%d", p2->no);
p2 = head;
max = 0;

while (1) {
  if (p2->priority > max && p2->active == 1)
    max = p2->no;
  p2 = p2->next;
  if (p2 == head)
    break;
}

printf("\n\tProcess with the id %d is the co-ordinator", max);

while (1) {
  printf("\nDo you want to continue?(y/n): ");
  flushall();
  scanf("%c", &ch);
  if (ch == 'n' || ch == 'N')
    break;
  p2 = head;

  while (1) {
    printf("\nEnter the process with id %d is active or not (0/1): ",
        p2->no);
    scanf("%d", &p2->active);
    p2 = p2->next;
    if (p2 == head)
      break;
  }

  printf("\nEnter the process id that invokes election algorithm: ");
  scanf("%d", &pid);

  printf("\n\tElection message is sent to processes ");

  while (p2->next != head) {
    if (p2->no > pid)
```

```
        printf("%d", p2->no);
      p2 = p2->next;
    }
    printf("%d", p2->no);
    p2 = head;
    max = 0;

    while (1) {
      if (p2->no > max && p2->active == 1)
        max = p2->no;
      p2 = p2->next;
      if (p2 == head)
        break;
    }
    printf("\n\tProcess with id %d is the co-ordinator", max);
  }
}

void main() {
  clrscr();
bully();
getch();
}
```

Conclusion : Hence we have studied and implemented Bully Election Algo successfully.

EXPERIMENT NO:5 DOP: DOS: GRADE:

TITLE OF EXPERIMENT: MUTUAL EXCLUSION

AIM: IMPLEMENTATION OF MUTUAL EXCLUSION ALGO

AIM: To study Implementation of Mutual Exclusion algorithms.

THEORY:

In computer science, **mutual exclusion** refers to the requirement of ensuring that no two concurrent processes are in their critical section at the same time; it is a basic requirement in concurrency control, to prevent race conditions. Here, a critical section refers to a period when the process accesses a shared resource, such as shared memory. The requirement of mutual exclusion was first identified and solved by Edsger W. Dijkstra in his seminal 1965 paper titled *Solution of a problem in concurrent programming control*, and is credited as the first topic in the study of concurrent algorithms.

Program Code:

```
#include <pthread.h>
 #include <stdio.h>
 int count = 0;
 pthread_mutex_t thread_lock;

 void* run_thread()
{
   pthread_mutex_lock(&thread_lock);
   pthread_t thread_id = pthread_self();
  printf("Thread %u: Current value of count = %d\n", thread_id, count);
   printf("Thread %u incrementing count ...\n");
   count++;
   sleep(1);
   printf("Value of count after incremented by thread %u = %d\n", thread_id, count);
   pthread_mutex_unlock(&thread_lock);
   pthread_exit(NULL);
 }

 int main (int argc, char *argv[])
 {
   pthread_t thread_array[4];
   int i = 0, ret, thread_num = 4;

   for (i = 0; i < thread_num; i++) {
     if ((ret = pthread_create(&thread_array[i], NULL, run_thread, NULL)==-1) {        printf("Thread
creation failed with return code: %d", ret);
        exit(ret);
    }
  }
   pthread_exit(NULL);}
```

Conclusion:

EXPERIMENT NO: 6 DOP: DOS: GRADE:

TITLE OF EXPERIMENT: MULTI-THREADED CLIENT-SERVER PROCESSES

AIM: WAP FOR MT CLIENT-SERVER PROCESSES.

AIM: To write a Program for multi-threaded client/server processes.
THEORY:

Multithreading is mainly found in multitasking operating systems. Multithreading is a widespread programming and execution model that allows multiple threads to exist within the context of a single process. These threads share the process's resources, but are able to execute independently. The threaded programming model provides developers with a useful abstraction of concurrent execution. Multithreading can also be applied to a single process to enable parallel execution on a multiprocessing system.

Program Code:
Server Side:

```
#include<stdio.h>
#include<string.h>    //strlen
#include<stdlib.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h>    //write
#include<pthread.h> //for threading , link with lpthread

//the thread function
void *connection_handler(void *);

int main(int argc , char *argv[])
{
   int socket_desc , client_sock , c , *new_sock;
   struct sockaddr_in server , client;

   //Create socket
   socket_desc = socket(AF_INET , SOCK_STREAM , 0);
   if (socket_desc == -1)
   {
      printf("Could not create socket");
   }
   puts("Socket created");

   //Prepare the sockaddr_in structure
   server.sin_family = AF_INET;
   server.sin_addr.s_addr = INADDR_ANY;
   server.sin_port = htons( 3000 );

   //Bind
   if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
   {
      //print the error message
```

```
    perror("bind failed. Error");
    return 1;
  }
  puts("bind done");

  //Listen
  listen(socket_desc , 3);

  //Accept and incoming connection
  puts("Waiting for incoming connections...");
  c = sizeof(struct sockaddr_in);

    c=sizeof(struct sockaddr_in);
   while(client_sock=accept(socket_desc,(struct sockaddr*)&client,(socklen_t*)&c))
   {
    puts("Connection accepted");

    pthread_t sniffer_thread;
    new_sock = malloc(1);
    *new_sock = client_sock;

    if( pthread_create( &sniffer_thread , NULL ,  connection_handler , (void*) new_sock) < 0)
    {
      perror("could not create thread");
      return 1;
    }

    puts("Handler assigned");
  }

  if (client_sock < 0)
  {
    perror("accept failed");
    return 1;
  }
  return 0;
}
/*
 This will handle connection for each client
 */
void *connection_handler(void *socket_desc)
{
  //Get the socket descriptor
  int sock = *(int*)socket_desc;
  int n;

    char    sendBuff[100], client_message[2000];

   while((n=recv(sock,client_message,2000,0))>0)
   {

    send(sock,client_message,n,0);
   }
```

```
    close(sock);

    if(n==0)
    {
      puts("Client Disconnected");
    }
    else
    {
      perror("recv failed");
    }
   return 0;
}
```

Client Side:

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>

#define MAX_SIZE 50

int main()
{
   int sock_desc;
   struct sockaddr_in serv_addr;
   char sbuff[MAX_SIZE],rbuff[MAX_SIZE];

   if((sock_desc = socket(AF_INET, SOCK_STREAM, 0)) < 0)
      printf("Failed creating socket\n");

   bzero((char *) &serv_addr, sizeof (serv_addr));

   serv_addr.sin_family = AF_INET;
   serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
   serv_addr.sin_port = htons(3000);

   if (connect(sock_desc, (struct sockaddr *) &serv_addr, sizeof (serv_addr)) < 0) {
      printf("Failed to connect to server\n");
      return -1;
   }

   printf("Connected successfully - Please enter string\n");
   while(fgets(sbuff, MAX_SIZE , stdin)!=NULL)
   {
    send(sock_desc,sbuff,strlen(sbuff),0);

        if(recv(sock_desc,rbuff,MAX_SIZE,0)==0)
         printf("Error");
        else
```

```
        fputs(rbuff,stdout);

      bzero(rbuff,MAX_SIZE);//to clean buffer-->IMP otherwise previous word characters also came
    }
       close(sock_desc);
    return 0;

}
```

Conclusion: Hence we have understood and run this multi-threaded client-server processes program successfully.

EXPERIMENT NO:7	**DOP:**	**DOS:**	**GRADE:**
TITLE OF EXPERIMENT: PROCESS/CODE MIGRATION			
AIM: WAP TO DEMONSTRATE PROCESS/CODE MIGRATION			

AIM: To write a Program to demonstrate process/code migration.
THEORY:
Introduction:

Program Code:

EXPERIMENT NO:8 DOP: DOS: GRADE:

TITLE OF EXPERIMENT: EJB DISTRIBUTED APPLICATION

AIM: WRITE A DISTRIBUTED APPLICATION USING EJB.

AIM: Write a distributed application using EJB.

THEORY:

Enterprise JavaBeans(EJB) is a specification for a component model that promises to simplify the development of multi-tier application systems capable of supporting high-volume business transactions [Spitzer 98]. EJB is not an implementation, but a specification owned by JavaSoft. JavaSoft is acting in the role of a standards organization to expedite the evolution of EJB technology.
EJB encourages innovation by allowing multiple vendors to develop different implementations of the specification. Most vendors add unique features to core application server functionality to differentiate themselves from their competitors. However, the EJB specification maintains that software developed in an EJB-compliant server can run in another EJB-compliant server seamlessly and without adaptation. In this paper, we examine Enterprise Bean portability among EJB-compliant servers and identify practical obstacles to portability.

Program Code:

```
package org.acme;

import java.rmi.RemoteException;
import javax.ejb.*;

public class HelloBean implements SessionBean {
 private SessionContext sessionContext;
 public void ejbCreate() {
 }
 public void ejbRemove() {
 }
 public void ejbActivate() {
 }
 public void ejbPassivate() {
 }
 public void setSessionContext(SessionContext sessionContext) {
  this.sessionContext = sessionContext;
 }
 public String sayHello() throws java.rmi.RemoteException {
  return "Hello World!!!!!";
 }
}
```

Conclusion:

EXPERIMENT NO: 9	**DOP:**	**DOS:**	**GRADE:**
TITLE OF EXPERIMENT: CORBA			
AIM: WAP USING CORBA TO DEMONSTRATE OBJECT BROKERING			

AIM: Write a program using CORBA to demonstrate object brokering.
THEORY:

OBJECTIVE (AIM) OF THE EXPERIMENT:

- To Create a Component for retrieving stock market exchange information using CORBA.

FACILITIES REQUIRED AND PROCEDURE:

a) Facilities Required:

S.No.	**Facilities required Quantity**
1	System 1
2	O/S Windows XP
3	S/W name JAVA

b) Procedure:

Step no.	**Details of the step**
1	Define the IDL interface
2	Implement the IDL interface using idlj compiler
3	Create a Client Program
4	Create a Server Program
5	Start orbed.
6	Start the Server.
7	Start the client

c) Program:

Define IDL Interface

```
modulesimplestocks{
interfaceStockMarket
{
floatget_price(in string symbol);
};
};
```

Note: Save the above module as simplestocks.idl
Compile the saved module using the idlj compiler as follows .
C:\WT\corba>idlj simplestocks.idl
After compilation a sub directory called simplestocks same as module name will be created and it generates the following files as listed below.
C:\WT\corba>idlj –fall simplestocks.idl

C:\WT\corba>cd simplestocks
C:\WT\corba\simplestocks>dir
Volume in drive C has no label.
Volume Serial Number is 348A-27B7
Directory of C:\suji\corba\simplestocks

02/06/2007 11:38 AM<DIR>

02/06/2007 11:38 AM <DIR> ..

02/06/2007 11:38 AM 2,071 StockMarketPOA.java
02/07/2007 02:15 PM 2,090 _StockMarketStub.java
02/07/2007 02:15 PM 865 StockMarketHolder.java
02/07/2007 02:15 PM 2,043 StockMarketHelper.java
02/07/2007 02:15 PM 359 StockMarket.java
02/07/2007 02:15 PM 339 StockMarketOperations.java
02/07/2007 02:08 PM 226 StockMarket.class
02/07/2007 02:08 PM 180 StockMarketOperations.class
02/07/2007 02:08 PM 2,818 StockMarketHelper.class
02/07/2007 02:08 PM 2,305 _StockMarketStub.class
02/06/2007 11:44 AM 2,223 StockMarketPOA.class
11 File(s) 15,519 bytes
2 Dir(s) 6,887,636,992 bytes free
C:\WT\corba\simplestocks>

// Implement the interface

```
importorg.omg.CORBA.*;
importsimplestocks.*;
public class StockMarketImpl extends StockMarketPOA{ private ORB orb;
public void setORB(ORB v){orb=v;}
public float get_price(String symbol) {
float price=0;
for(int i=0;i<symbol.length();i++){
price+=(int)symbol.charAt(i);}
price/=5;
return price;}
publicStockMarketImpl(){super();}}
```

//Server Program:

```
importorg.omg.CORBA.*;
importorg.omg.CosNaming.*;
importorg.omg.CosNaming.NamingContextPackage.*;
importorg.omg.PortableServer.*;
importorg.omg.PortableServer.POA.*;
importjava.util.Properties;
importsimplestocks.*;
public class StockMarketServer{
public static void main(String[] args) {
try { ORB orb=ORB.init(args,null);
POA rootpoa=POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
rootpoa.the_POAManager().activate();
StockMarketImplss=new StockMarketImpl();
```

```
ss.setORB(orb);
org.omg.CORBA.Object ref=rootpoa.servant_to_reference(ss);
StockMarkethrf=StockMarketHelper.narrow(ref);
org.omg.CORBA.Objectorf=orb.resolve_initial_references("NameService");
NamingContextExtncrf=NamingContextExtHelper.narrow(orf);
NameComponentpath[]=ncrf.to_name("StockMarket");

ncrf.rebind(path,hrf);

System.out.println("StockMarket server is ready");
//Thread.currentThread().join();
orb.run();}catch(Exception e){
e.printStackTrace();}}}
```

// Client Program:

```
importorg.omg.CORBA.*;
importorg.omg.CosNaming.*;
importsimplestocks.*;
importorg.omg.CosNaming.NamingContextPackage.*;
public class StockMarketClient{
public static void main(String[] args) {
try
{
ORB orb=ORB.init(args,null);
NamingContextExt
ncRef=NamingContextExtHelper.narrow(orb.resolve_initial_references("NameService"))
//NameComponentpath[]={new NameComponent("NASDAQ","")};
StockMarket market=StockMarketHelper.narrow(ncRef.resolve_str("StockMarket"));
System.out.println("Price of My company is"+market.get_price("My_COMPANY"));}
catch(Exception e){
e.printStackTrace();}}}
Compile the above files as
C:\WT\corba>javac *.java

C:\WT\corba>start orbd -ORBInitialPort 1050 -ORBInitialHostlocalhost

C:\WT\corba>start java StockMarketServer -ORBInitialPort 1050 -ORBInitialHost
localhost
C:\WT\corba>
StockMarket server is ready
C:\WT\corba>java StockMarketClient -ORBInitialPort 1050 –ORBInitialHostlocalhost
```

d) Output:

Server Side:

```
D:\>cd MWT\corbastock
D:\MWT\CorbaStock>set path="c:\j2sdk1.4.1\bin";
D:\MWT\CorbaStock>idlj simplestocks.idl
D:\MWT\CorbaStock>idlj -fall simplestocks.idl
D:\MWT\CorbaStock>javac *.java
D:\MWT\CorbaStock>start orbd -ORBInitialPort 1050 -ORBInitialHostlocalhost
```

D:\MWT\CorbaStock>java StockMarketServer -ORBInitialPort 1050 -ORBInitialHostlocalhost
StockMarket server is ready

Client Side:
D:\MWT\CorbaStock>java StockMarketClient -ORBInitialPort 1050 -ORBInitialHostlocalhost
Price of My Company is: 165.6
D:\MWT\CorbaStock>

Conclusion:
Thus the above program is used to develop a component for retrieving stock market exchange information using CORBA and it is executed successfully.

EXPERIMENT NO: 10 DOP: DOS: GRADE:

TITLE OF EXPERIMENT: DCOM/.NET

AIM: DEPLOY A DISTRIBUTED APPLICATION USING .NET FRAMEWORK

AIM: Use .Net framework to deploy a distributed application.
THEORY:

Introduction:
Program Code:
Develop a component to retrieve Message Box Information Using DCOM/.NET:

OBJECTIVE (AIM) OF THE EXPERIMENT:

- To create a component to retrieve message box information using DCOM/.NET

FACILITIES REQUIRED AND PROCEDURE

a) Facilities Required:

S.No.	Facilities required Quantity
1.	System 1
2.	O/S Windows XP
3.	S/W name Microsoft Visual Studio .Net

b) Procedure:

Step no.	Details of the step
PART I	
1.	Start the process.
2.	Open Visual Studio. NET.
3.	Goto File->New->Project->ClassLibrary|Empty Library->OK
4.	Goto Solution Explorer->Right Click->Add->Add Component|Add New Item->COM Class-_OK
5.	Add the following codings Save & Build.

PART II

1.

Go To Start->Microsoft Visual.Net 2003->Visual Stufio.Net tools->Command prompt Setting environment for using Microsoft Visual Studio .NET 2003 tools. (If you have another version of Visual Studio or Visual C++ installed and wish to use its tools from the command line, run vcvars32.bat for that version.) C:\Documents and Settings\administrator>sn -k ms.snk
Microsoft (R) .NET Framework Strong Name Utility Version 1.1.4322.573 Copyright (C) Microsoft Corporation 1998-2002. All rights reserved. Key pair written to ms.snk C:\Document and Settings\administrator> Copy ms.snk to bin directory (locate the class library)

2.
start -> settings -> control panel->administrative tools->component services-> computer->my computer->com + Application -> new ->application ->next -> create an empty application-> choose the server Application -> enter the new Application name (mssg) -> next ->choose the interactive user-> next->finish.
3.
expandmssg -> click the components ->right click -> new-> component - >next->install new event classes-> select the class library1.tlb(class library->bin->open->next->finish.

PART III

1. Open Visual studio .net -> file-> new ->Project->Windows Application
2. Create one label box,one text box and one button in the form.
3. Include the following code in the Button click event
4. execute the project

c) Program:

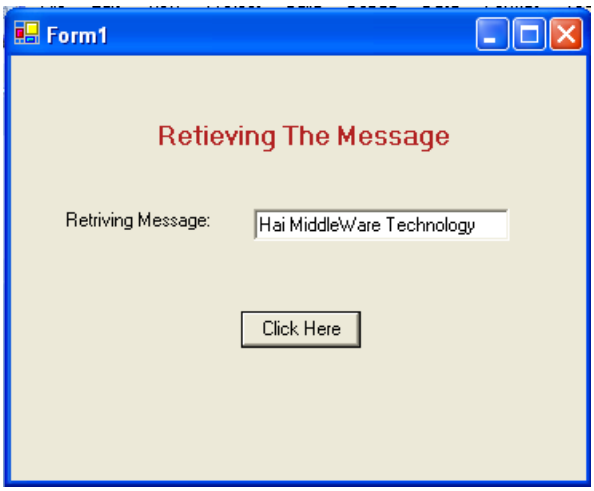
PART-I

```
Public Function test () As String
Dim str = "HaiMiddleWare Technology"
Return (str)
End Function
Public Function create () As String
MsgBox(test())
End Function
```

PART-III

```
Imports msg
Public Class Form1
Inherits System.Windows.Forms.Form
Dim mo As New msg.ComClass1
Private Sub Button1_Click (ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
textbox1.text = mo.test()
End Sub
End Class
```

d) Output:



Conclusion:
Thus the above program is used to develop a component to retrieve message box information using DCOM/.Net and it is executed successfully.