SOFTWARE REQUIREMENT :

1. ~~Turbo C++ IDE (TurboC3)~~
2. ~~Borland Turbo C++ (Version 4.5)~~
3. JAVA

REFERENCE BOOKS :

1. Tanenbaum, Andrew. S., Distributed Operation System, PHI.
2. Distributed Systems: Concepts and Design Edition 4, by George Coulouris, Jean Dollimore and Tim Kindberg
3. P K Sinha, “Distributed operating systems; Concepts and design”, PHI Learning.
4. Singhal & Shivaratri, "Advanced Concept in Operating Systems", McGraw Hill

LIST OF PROGRAMS

1. Case study on Common Object Request Broker Architecture.

2. Implementation of Deadlock through Simulation.

3. Study of 3 tier client server architecture.

4. Case study on Client and RMI Server.

5. WAP to Implement an Election algorithm.

6. S/W Simulation for Clock Synchronization in Distributed System using Lamport's Algorithm.

7. Implementation of Banker's Algorithm for avoiding Deadlock

8. Case study on Inventory Management

9. Case study on Supply Chain Management:

10. Case study on Reservation System:

PROGRAM NO: - 1

PROBLEM DEFINITION:

Case study on Common Object Request Broker Architecture.

The Common Object Request Broker Architecture (CORBA) for ORB technology and Java applet technology for Web technology. We areusing exemplars to make the discussion more concrete. We are using these particular exemplarsbecause they are the most visible and popular representatives of their respective classes.While this decision may somewhat color our discussion of DOT with the peculiarities ofCORBA and Java, our intent is to address the broader aspects of DOT.To provide the necessary perspective, we will give a brief explanation of the origins and currentstates of these exemplars. They arose from two broad classes of DOT progenitors: Operatingsystems and distributed systems infrastructures influenced CORBA, and programming languagesand the Web gave way to Java.

As object technology became more popular through the 1980s there was more interest in bundling the concept of objects with the concept of transparent distributed computing. Objects,with their inherent combination of data and behavior and their strict separation of interfacefrom implementation, offer an ideal package for distributing data and processes to end-userapplications. Objects became an enabling technology for distributed processing. In the early1990s an international trade association called the Object Management Group (OMG) [OMG 97] defined a standard for the distribution of objects. The OMG defined the Common ObjectRequest Broker Architecture (CORBA), which provided a standard by which OT could be usedin distributed computing environments. The latest version of this standard, CORBA 2.0, addressesissues related to interface, registration, databases, communication, and error handling.When combined with other object services defined by the OMG Object ManagementArchitecture (OMA), CORBA becomes a middleware that facilitates full exploitation of objecttechnology in a distributed system. However, if we were to characterize CORBA technology inthe simplest possible language, it would be to say it is an object-oriented RPC.

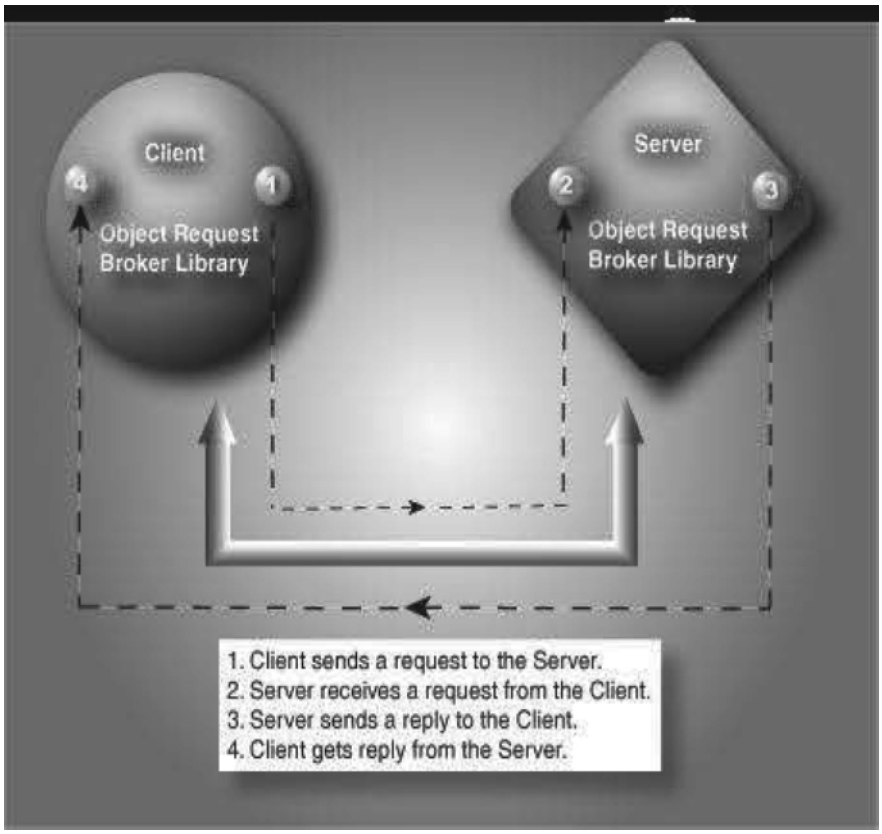


Fig. CORBA Archtitecture

PROGRAM NO: - 2

PROBLEM DEFINITION:

Implementation of Deadlock through Simulation.

```
#include
#include     //We are using rand() function:)
int main()
{
    int alloc[10][10],req[10][10],ins[10],avail[10],tp,tr,i,j;
    int tmp[10]={0},count=0;
    bool finish[10]={false},flag;
    printf("Enter total no of processes:");
    scanf("%d",&tp);
    printf("Enter total no of resources:");
    scanf("%d",&tr);
    printf("Randomly Generated Allocation Matrix:\n");     //itz tp x
tr order matrix
    for(i=0;i
    {
        for(j=0;j
        {
            alloc[i][j]=rand()%5;
            printf("\t%d",alloc[i][j]);
        }
        printf("\n");
    }
    printf("Randomly Generated Request Matrix:\n");
    for(i=0;i
    {
        for(j=0;j
        {
            req[i][j]=rand()%5;
            printf("\t%d",req[i][j]);
        }
        printf("\n");
    }
    printf("Randomly Generated Resource Instance Vetctor:\n");
    for(i=0;i
    {
        ins[i]=rand()%10+tr+tp;       //Just to increase resource
instances
        printf("\t%d",ins[i]);
    }

    //To calculate resource availability vector

    for(i=0;i
    {
        for(j=0;j
        {
            tmp[i]+=alloc[j][i];     //sum calculated columnwise for
allocation matrix
        }
    }
    printf("\nCalculated Availability Vector:\n");
    for(i=0;i
    {
        avail[i]=ins[i]-tmp[i];
        printf("\t%d",avail[i]);
    }
```

```
    //main logic starts:P
    while(count
    {                               //if finish array has all
true's(all processes to running state)
                                    //deadlock not detected and loop
stops!
    for(i=0;i
    {
        count=0;
        //To check whether resources can be allocated any to blocked
process
        if(finish[i]==false)
        {
            for(j=0;j
            {
                    if(req[i][j]<=avail[j])
                    {
                        count++;
                    }
            }
            flag=false;
            if(count==tr)
            {
                for(j=0;j
                {
                    avail[j]+=alloc[i][j];        //allocated
reources are released and added to available!
                }
                finish[i]=true;
                printf("\nProcess %d is transferred to running state
and assumed finished",i+1);
            }
            else
                flag=true;
        }
    }
    count=0;
    for(j=0;j
    {
        if(finish[j]==true)
        {
            count++;
        }
    }
    }
    for(i=0;i
    {
        if(finish[i]==false)
        {
            printf("\n Oops! Deadlock detected and causing process
is:process(%d)\n",i+1);
            break;
        }
    }
    i=i-1;
    if(finish[i]==true)
    printf("\nHurray! Deadlock not detected:-)\n");
    return 0;
}
```

Output:

Enter total no of processes:7
Enter total no of resources:3
Randomly Generated Allocation Matrix:

3 1 2
0 3 0
1 2 4
1 2 2
0 4 3
1 0 1
2 1 1

Randomly Generated Request Matrix:

3 2 4
2 0 2
3 2 0
4 2 2
3 4 2
3 1 1
2 4 3

Randomly Generated Resource Instance Vetctor:

11 19 14

Calculated Availability Vector:

3 6 1

Process 3 is transferred to running state and assumed finished
Process 4 is transferred to running state and assumed finished
Process 5 is transferred to running state and assumed finished
Process 6 is transferred to running state and assumed finished
Process 7 is transferred to running state and assumed finished
Process 1 is transferred to running state and assumed finished
Process 2 is transferred to running state and assumed finished

PROGRAM NO: - 3

PROBLEM DEFINITION:

Study of 3 tier client server architecture

OBJECTIVE:

To revise concept the standard 3-tier architecture consists of presentation and application logic in the client, application and business logic in a middle tier application server, and data managed by database servers in the 3 tier.

Architecture Diagram:

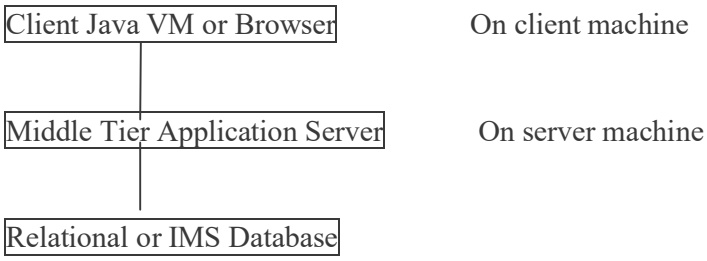


FIGURE 1 TIER ARCHITECTURE

Three-tier architecture is a client–server architecture in which the user interface, functional process logic ("business rules"), computer data storage and data access are developed and maintained as independent modules, most often on separate platforms.[3] It was developed by John J. Donovan in Open Environment Corporation (OEC), a tools company he founded in Cambridge, Massachusetts. The three-tier model is a software architecture pattern. Apart from the usual advantages of modular software with well-defined interfaces, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology. For example, a change of operating system in the presentation tier would only affect the user interface code.

Typically, the user interface runs on a desktop PC or workstation and uses a standard graphical user interface, functional process logic that may consist of one or more separate modules running on a workstation or application server, and an RDBMS on a database server or mainframe that contains the computer data storage logic. The middle tier may be multi-tiered itself (in which case the overall architecture is called an "n-tier architecture").

N-tier application architecture provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application. Three-tier architecture is typically composed of a presentation tier, a business or data access tier, and a data tier. While the concepts of layer and tier are often used interchangeably, one fairly common point of view is that there is indeed a difference. This view holds that a layer is a logical structuring mechanism for the elements that make up the software solution, while a tier is a physical structuring mechanism for the system infrastructure

PROGRAM NO: - 4

PROBLEM DEFINITION:

Case study on Client and RMI Server

In this section, you will learn how to send massage from RmiClient to the RmiServer. Here, we are going to create "ReceiveMessageInterface" interface. The interface defines the methods that can be invoked from the client. Essentially, the interface defines the client's view of the remote object. After that, we will create a class named "RMIServer". The RMI Server accepts tasks from clients, runs the tasks, and returns any result. The server code consists of an interface and a class.

In this class, the "receiveMessage()" method, which is called from the remote client, is defined. This class is the implementation of the RMI interface. The RmiServer creates the "registry". This is a kind of directory. Its key is a name (which is the ID of a remote object) and its content is an object. This object is looked up from a remote program by the name. This registry is accessed from a remote object by the IP address or host name and the port number.

createRegistry(): This is the method creates and exports a registry on the local host that accepts requests on the specified port.

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.

The **Java Remote Method Invocation** (**Java RMI**) is a Java API that performs the object-oriented equivalent of remote procedure calls (RPC), with support for direct transfer of serialized Java objects and distributed garbage collection.

1. The original implementation depends on Java Virtual Machine (JVM) class representation mechanisms and it thus only supports making calls from one JVM to another. The protocol underlying this Java-only implementation is known as Java Remote Method Protocol (JRMP).
2. In order to support code running in a non-JVM context, a CORBA version was later developed.

Usage of the term **RMI** may denote solely the programming interface or may signify both the API and JRMP, whereas the term RMI-IIOP (read: RMI over IIOP) denotes the RMI interface delegating most of the functionality to the supporting CORBA implementation.

PROGRAM NO: - 5

PROBLEM DEFINITION:

WAP to Implement an Election algorithm.

```
#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 p;

struct priority
{
     int pri;
     struct priority *next;
     p *pp;
};

typedef struct priority pri;

pri* find_priority(p *head,pri *head1)
{
p *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()
{
p *head;
p *p1;
p *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("nnEnter 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=(p*)malloc(sizeof(p));
                  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=(p*)malloc(sizeof(p));
```

```
                    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("nnProcess with id %d has invoked election algorithm",pid);
printf("tnnElection 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("ntProcess with the id %d is the co-ordinator",max);
while(1)
       {
           printf("nnDo you want to continue?(y/n): ");
           flushall();
           scanf("%c",&ch);
           if(ch=='n' || ch=='N')
                  break;
           p2=head;

           while(1)
              {
              printf("nnEnter 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("nnEnter the process id that invokes election
algorithm: ");
           scanf("%d",&pid);
           printf("ntElection 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("nntProcess with id %d is the co-ordinator",max);
       }
}

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

Output:

```
 enter the number of proceess   4

enter the name of process       p1

enter the priority of process   4

enter the name of process       p2

enter the priority of process   3

enter the name of process       p3

enter the priority of process   6

enter the name of process       p4

enter the priority of process   1

p3      6
p1      4
p2      3
p4      1
process p3 select aas coordinator
1)election
 2) exit        1

 1)intialise election   p2

process p2 send message to p3
process p2 send message to p1
process p1 send OK message to p2
process p1 send message to p3
process p1 is select as new coordinator
process p1 send alert message to p2
process p1 send alert message to p4
```

PROGRAM NO: - 6

PROBLEM DEFINITION:

S/W Simulation for Clock Synchronization in Distributed System using Lamport's Algorithm.

```
#include<conio.h>
int max1(int a, int b)    //to find the maximum timestamp between two events
{
   if (a>b)
   return a;
else
return b;
}
int main()
{
int i,j,k,p1[20],p2[20],e1,e2,dep[20][20];
printf("enter the events : ");
scanf("%d %d",&e1,&e2);
for(i=0;i<e1;i++)
p1[i]=i+1;
for(i=0;i<e2;i++)
p2[i]=i+1;
printf("enter the dependency matrix:\n");
printf("\t enter 1 if e1->e2 \n\t enter -1, if e2->e1 \n\t else enter 0 \n\n");
for(i=0;i<e2;i++)
printf("\te2%d",i+1);
for(i=0;i<e1;i++)
{
printf("\n e1%d \t",i+1);
for(j=0;j<e2;j++)
scanf("%d",&dep[i][j]);
}

for(i=0;i<e1;i++)
{
   for(j=0;j<e2;j++)
   {
      if(dep[i][j]==1)     //change the timestamp if dependency exist
      {    p2[j]=max1(p2[j],p1[i]+1);
         for(k=j;k<e2;k++)
         p2[k+1]=p2[k]+1;
      }
      if(dep[i][j]==-1)    //change the timestamp if dependency exist
      {
         p1[i]=max1(p1[i],p2[j]+1);
         for(k=i;k<e1;k++)
         p2[k+1]=p1[k]+1;
      }

   }
```

```
}
printf("P1 : ");     //to print the outcome of Lamport Logical Clock
for(i=0;i<e1;i++)
{
printf("%d",p1[i]);
}
printf("\n P2 : ");
for(j=0;j<e2;j++)
printf("%d",p2[j]);

getch();
return 0 ;
}
```

OUTPUT-

```
Enter the number of process:
2
Enter the no of events per process:
7
5
Enter the relationship:
For process:1
For event:1
0
For event:2
0
For event:3
0
For event:4
0
For event:5
22
For event:6
0
For event:7
24
For process:2
For event:1
0
For event:2
0
For event:3
12
For event:4
0
For event:5
16
```

PROGRAM NO: - 7

PROBLEM DEFINITION:

Implementation of Banker's Algorithm for avoiding Deadlock

```
#include<stdio.h>
#include<conio.h>
void main()
{
int clm[7][5],req[7][5],alloc[7][5],rsrc[5],avail[5],comp[7];
int first,p,r,i,j,prc,count,t;
clrscr();
count=0;
for(i=1;i<=7;i++)
comp[i]=0;
printf("Enter the no of processes:\n");
scanf("%d",&p);
printf("Enter the no of resources:\n");
scanf("%d",&r);
printf("Enter the claim for each process:");
for(i=1;i<=p;i++)
{
printf("\nFor process %d",i);
for(j=1;j<=r;j++)
{
scanf("%d",&clm[i][j]);
}
}
printf("Enter the allocation for each process:\n");
for(i=1;i<=p;i++)
{
printf("\nFor process ",i);
for(j=1;j<=r;j++)
{
scanf("%d",&alloc[i][j]);
}
}
printf("Enter total no of each resource:");
for(j=1;j<=r;j++)
scanf("%d",&rsrc[j]);
for(j=1;j<=r;j++)
{
int total=0;
avail[j]=0;
for(i=1;i<=p;i++)
{total+=alloc[i][j];}
avail[j]=rsrc[j]-total;
}
do
```

```
{
for(i=1;i<=p;i++)
{
for(j=1;j<=r;j++)
{
req[i][j]=clm[i][j]-alloc[i][j];
}
}
printf("\n\nAvailable resorces is:");
for(j=1;j<=r;j++)
{ printf(" ",avail[j]); }
printf("\nClaim matrix:\t\tAllocation matrix:\n");
for(i=1;i<=p;i++)
{
for(j=1;j<=r;j++)
{
printf("%d",clm[i][j]);
}
printf("\t\t\t");
for(j=1;j<=r;j++)
{
printf("%d",alloc[i][j]);
}
printf("\n");
}
prc=0;
for(i=1;i<=p;i++)
{
if(comp[i]==0)//if not completed
{
prc=i;
for(j=1;j<=r;j++)
{
if(avail[j]
{
prc=0;
break;
}
}
}
if(prc!=0)
break;
}
if(prc!=0)
{
printf("\nProcess ",prc,"runs to completion!");
count++;
for(j=1;j<=r;j++)
{
avail[j]+=alloc[prc][j];
alloc[prc][j]=0;
clm[prc][j]=0;
comp[prc]=1;
}
}
```

```
}
while(count!=p&&prc!=0);
if(count==p)
printf("\nThe system is in a safe state!!");
else
printf("\nThe system is in an unsafe state!!");
getch();
}
```

OUT PUT:

Enter the no of processes:
2
Enter the no of resources:
3

Enter the claim for each process:

For process I : 2 4 5

For process II: 2 5 3

Enter the total no of each resource : 5 5 2

Available resource is :

Claim matrix: allocation matrix:
245 123
253 234

The system is in an unsafe state!!

PROGRAM NO: - 8

PROBLEM DEFINITION:

Case study on Inventory Management

Inventory or stock refers to the goods and materials that a business holds for the ultimate purpose of resale (or repair).

Inventory management is a science primarily about specifying the shape and percentage of stocked goods. It is required at different locations within a facility or within many locations of a supply network to precede the regular and planned course of production and stock of materials.

The scope of inventory management concerns the fine lines between replenishment lead time, carrying costs of inventory, asset management, inventory forecasting, inventory valuation, inventory visibility, future inventory price forecasting, physical inventory, available physical space for inventory, quality management, replenishment, returns and defective goods, and demand forecasting. Balancing these competing requirements leads to optimal inventory levels, which is an on-going process as the business needs shift and react to the wider environment.

Inventory management involves a retailer seeking to acquire and maintain a proper merchandise assortment while ordering, shipping, handling, and related costs are kept in check. It also involves systems and processes that identify inventory requirements, set targets, provide replenishment techniques, report actual and projected inventory status and handle all functions related to the tracking and management of material. This would include the monitoring of material moved into and out of stockroom locations and the reconciling of the inventory balances. It also may include ABC analysis, lot tracking, cycle counting support, etc. Management of the inventories, with the primary objective of determining/controlling stock levels within the physical distribution system, functions to balance the need for product availability against the need for minimizing stock holding and handling costs.

PROGRAM NO: - 9

PROBLEM DEFINITION:

Case studay on Supply Chain Management:

Supply chain management is a cross-functional approach that includes managing the movement of raw materials into an organization, certain aspects of the internal processing of materials into finished goods, and the movement of finished goods out of the organization and toward the end consumer. As organizations strive to focus on core competencies and becoming more flexible, they reduce their ownership of raw materials sources and distribution channels. These functions are increasingly being outsourced to other firms that can perform the activities better or more cost effectively. The effect is to increase the number of organizations involved in satisfying customer demand, while reducing managerial control of daily logistics operations. Less control and more supply chain partners led to the creation of the concept of supply chain management. The purpose of supply chain management is to improve trust and collaboration among supply chain partners, thus improving inventory visibility and the velocity of inventory movement.

The term "supply chain management" entered the public domain when Keith Oliver, a consultant at used it in an interview for the Financial Times in 1982. The term was slow to take hold. It gained currency in the mid-1990s, when a flurry of articles and books came out on the subject. In the late 1990s it rose to prominence as a management buzzword, and operations managers began to use it in their titles with increasing regularity.

Commonly accepted definitions of supply chain management include:

- The management of upstream and downstream value-added flows of materials, final goods, and related information among suppliers, company, resellers, and final consumers

- The systematic, strategic coordination of traditional business functions and tactics across all business functions within a particular company and across businesses within the supply chain, for the purposes of improving the long-term performance of the individual companies and the supply chain as a whole

- A customer-focused definition is given by Hines (2004:p76): "Supply chain strategies require a total systems view of the links in the chain that work together efficiently to create customer satisfaction at the end point of delivery to the consumer. As a consequence, costs must be lowered throughout the chain by driving out unnecessary expenses, movements, and handling. The main focus is turned to efficiency and added value, or the end-user's perception of value. Efficiency must be increased, and bottlenecks removed. The measurement of performance focuses on total system efficiency and the equitable monetary reward distribution to those within the supply chain. The supply chain system must be responsive to customer requirements."

PROGRAM NO: - 10

PROBLEM DEFINITION:

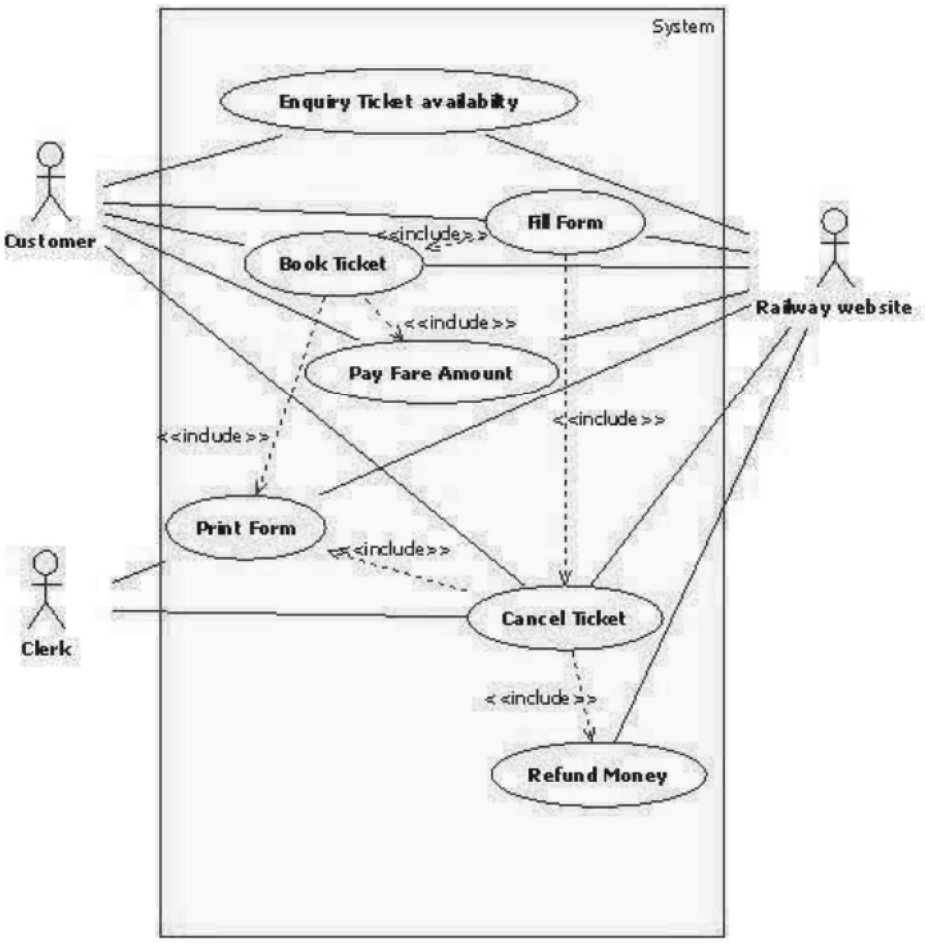
Case study on Reservation System:

Railway Reservation System is a system used for booking tickets over internet. Any Customer Can book tickets for different trains. Customer can book a ticket only if the tickets are available. Customer searches for the availability of tickets then if the tickets are available he books the tickets by initially filling details in a form. Tickets can be booked in two ways by i-ticket or by e-ticket booking.

In case of i-ticket booking customer can book the tickets online and the tickets are couriered to Particular customer at their address. But in case of e-ticket booking and cancelling tickets are booked and cancelled online sitting at the home and customer himself has to take print of the ticket but in both the cases amount for tickets are deducted from customers account. For cancellation of ticket the customer has to go at reservation office than fill cancellation form and ask the clerk to cancel the ticket than the refund is transferred to customer account. After booking ticket the customer has to check out by paying fare amount to The Customers are required to register on the server for getting access to the database and query result retrieval. Upon registration, each user has an account which is essentially the 'view level' for the customer. The account contains comprehensive information of the user entered during registration and permits the customer to get access to his past reservations, enquire about travel fare and availability of seats, make afresh reservations, update his account details, etc.

The Railway Administrator is the second party in the transactions. The administrator is required to login using a master password, once authenticated as an administrator, one has access and right of modification to all the information stored in the database at the server. This includes the account information of the customers, attributes and statistics of stations, description of the train stoppages and physical description of coaches, all the reservations that have been made, etc. The railway administrator has the right to modify any information stored at the server Database.

clerk.


System
Enquiry Ticket availability
Customer
Fill Form
<<include>>
Book Ticket
Railway website
<<include>>
Pay Fare Amount
<<include>>
<<include>>
Print Form
<<include>>
Clerk
Cancel Ticket
<<include>>
Refund Money