Name: Vishal Shashikant Salvi.

UID: 2019230069

Batch: C

Class: SE Comps

Experiment No. 5

Aim: To implement Bankers Algorithms.

Theory:

What is Banker's Algorithm?

Banker's Algorithm is used majorly in the banking system to avoid deadlock. It helps you to identify whether a loan will be given or not.

This algorithm is used to test for safely simulating the allocation for determining the maximum amount available for all resources. It also checks for all the possible activities before determining whether allocation should be continued or not.

For example, there are X number of account holders of a specific bank, and the total amount of money of their accounts is G.

When the bank processes a car loan, the software system subtracts the amount of loan granted for purchasing a car from the total money (G+ Fixed deposit + Monthly Income Scheme + Gold, etc.) that the bank has.

It also checks that the difference is more than or not G. It only processes the car loan when the bank has sufficient money even if all account holders withdraw the money G simultaneously.

Let us assume that there are n processes and m resource types. Some data structures that are used to implement the banker's algorithm are:

1. Available

It is an **array** of length m. It represents the number of available resources of each type. If Available[j] = k, then there are k instances available, of resource type R(j).

2. Max

It is an n x m matrix which represents the maximum number of instances of each resource that a process can request. If Max[i][j] = k, then the process P(i) can request atmost k instances of resource type R(j).

3. Allocation

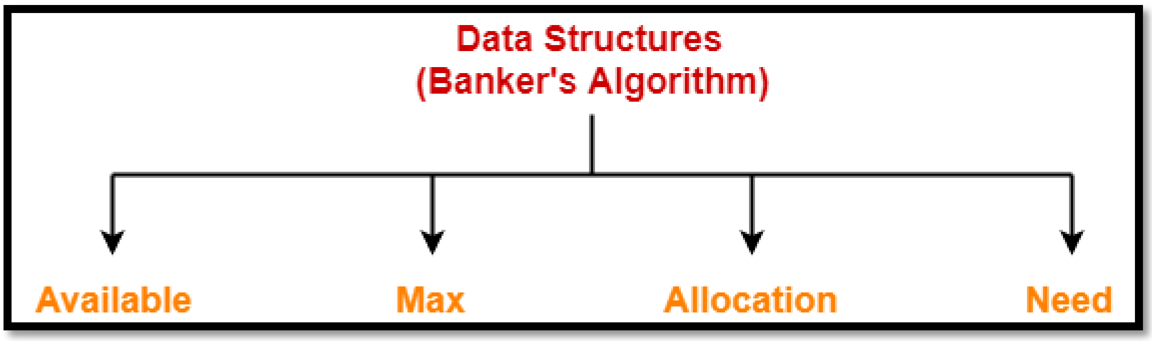
It is an n x m matrix which represents the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process P(i) is currently allocated k instances of resource type R(j).

4. Need

It is an n x m matrix which indicates the remaining resource needs of each process. If Need[i][j] = k, then process P(i) may need k more instances of resource type R(j) to complete its task.

Data Structures Used-

To implement banker's algorithm, following four data structures are used-



1. Available
2. Max
3. Allocation
4. Need

Data Structure	Definition	Example
Available	It is a single dimensional array that specifies the number of instances of each resource type currently available.	Available[R1] = K It means K instances of resource type R1 are currently available.
Max	It is a two dimensional array that specifies the maximum number of instances of each resource type that a process can request.	Max[P1][R1] = K It means process P1 is allowed to ask for maximum K instances of resource type R1.
Allocation	It is a two dimensional array that specifies the number of instances of each resource type that has been allocated to the process.	Allocation[P1][R1] = K It means K instances of resource type R1 have been allocated to the process P1.
Need	It is a two dimensional array that specifies the number of instances of each resource type that a process requires for execution.	Need[P1][R1] = K It means process P1 requires K more instances of resource type R1 for execution.

Characteristics of Banker's Algorithm

Here are important characteristics of banker's algorithm:

- Keep many resources that satisfy the requirement of at least one client
- Whenever a process gets all its resources, it needs to return them in a restricted period.
- When a process requests a resource, it needs to wait
- The system has a limited number of resources
- Advance feature for max resource allocation

Disadvantage of Banker's algorithm

Here, are cons/drawbacks of using banker's algorithm

- Does not allow the process to change its Maximum need while processing
- It allows all requests to be granted in restricted time, but one year is a fixed period for that.
- All processes must know and state their maximum resource needs in advance.

Summary:

- Banker's algorithm is used majorly in the banking system to avoid deadlock. It helps you to identify whether a loan will be given or not.
- Notations used in banker's algorithms are 1) Available 2) Max 3) Allocation 4) Need
- Resource request algorithm enables you to represent the system behavior when a specific process makes a resource request.
- Banker's algorithm keeps many resources that satisfy the requirement of at least one client
- The biggest drawback of banker's algorithm this that it does not allow the process to change its Maximum need while processing.

<u>Problem-01:</u>

A single processor system has three resource types X, Y and Z, which are shared by three processes. There are 5 units of each resource type. Consider the following scenario, where the column alloc denotes the number of units of each resource type allocated to each process, and the column request denotes the number of units of each resource type requested by a process in order to complete execution. Which of these processes will finish LAST?

1. P0
2. P1
3. P2
4. None of the above since the system is in a deadlock

	Allocated			**Request**		
	X	**Y**	**Z**	**X**	**Y**	**Z**
P0	1	2	1	1	0	3
P1	2	0	1	0	1	2

P2	2	2	1	1	2	0

Solution-

According to question-

- Total = [X Y Z] = [5 5 5]
- Total _Alloc = [X Y Z] = [5 4 3]

Now,

Available

= Total – Total_Alloc

= [5 5 5] – [5 4 3]

= [0 1 2]

Step-01:

- With the instances available currently, only the requirement of the process P1 can be satisfied.
- So, process P1 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Then,

Available

= [0 1 2] + [2 0 1]

= [2 1 3]

Step-02:

- With the instances available currently, only the requirement of the process P0 can be satisfied.
- So, process P0 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [2 1 3] + [1 2 1]

= [3 3 4]

Step-03:

- With the instances available currently, the requirement of the process P2 can be satisfied.
- So, process P2 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [3 3 4] + [2 2 1]

= [5 5 5]

Thus,

- There exists a safe sequence P1, P0, P2 in which all the processes can be executed.
- So, the system is in a safe state.
- Process P2 will be executed at last.

Thus, Option (C) is correct.

Code:

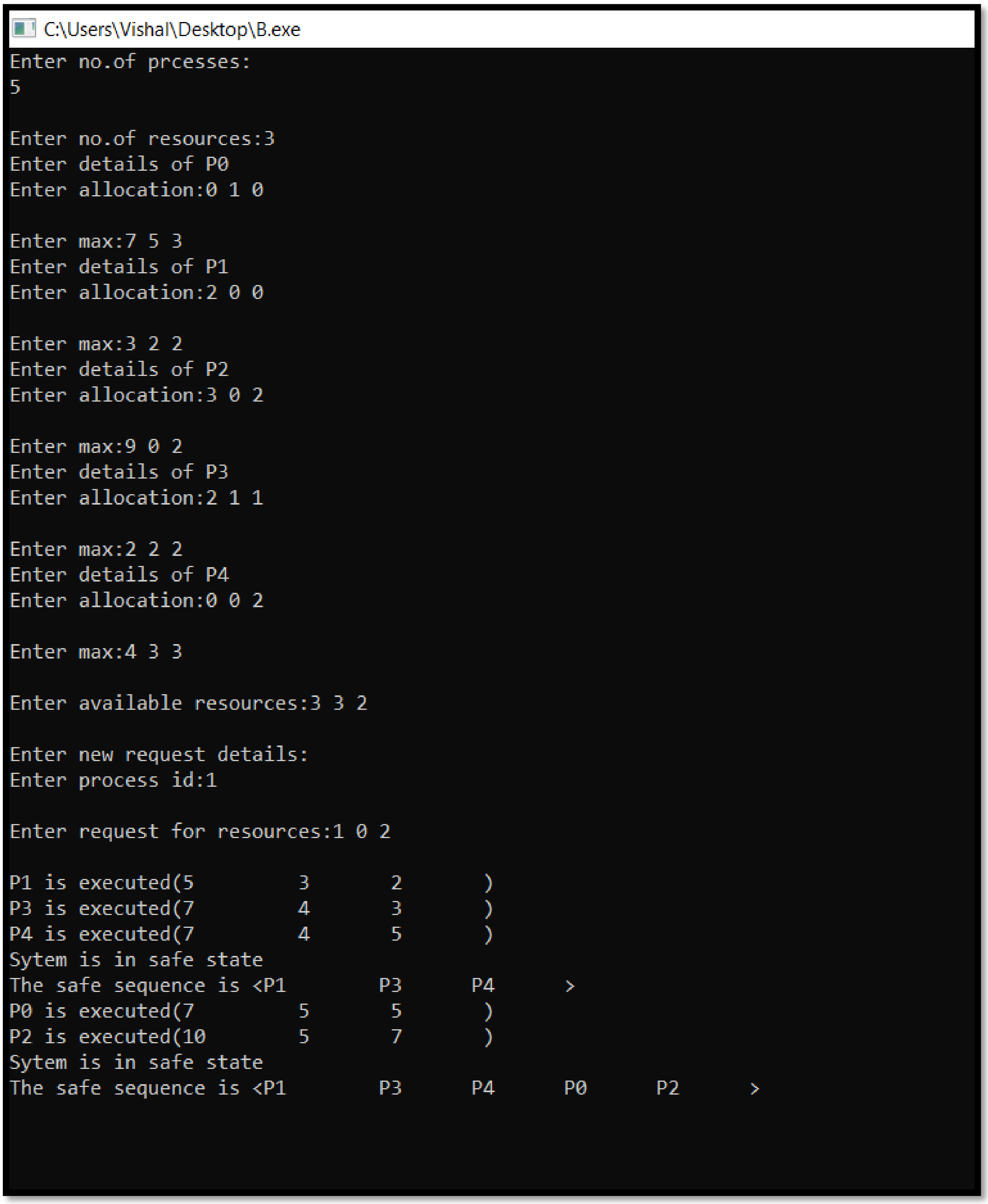
```
#include<stdio.h>
#include<conio.h>
struct file
{
        int alloc[10];
        int max[10];
        int need[10];
        int flag;
};
int main()
{
        struct file f[10];
        int fl;
        int i,j,k,n,r,id,newr;
        int b,g,p,count=0;
        int avail[10],seq[10];
        printf("Enter no.of prcesses:\n");
        scanf("%d",&n);
        printf("\nEnter no.of resources:");
        scanf("%d",&r);
        for(i=0;i<n;i++)
        {
                printf("Enter details of P%d",i);
        printf("\nEnter allocation:");
        for(j=0;j<r;j++)
                scanf("%d",&f[i].alloc[j]);
        printf("\nEnter max:");
        for(j=0;j<r;j++)
                scanf("%d",&f[i].max[j]);
        f[i].flag=0;
}
printf("\nEnter available resources:");
for(i=0;i<r;i++)
        scanf("%d",&avail[i]);
printf("\nEnter new request details:");
printf("\nEnter process id:");
scanf("%d",&id);
printf("\nEnter request for resources:");
for(i=0;i<r;i++)
{
        scanf("%d",&newr);
        f[id].alloc[i]+=newr;
        avail[i]=avail[i]-newr;
}
for(i=0;i<n;i++)
{
                for(j=0;j<r;j++)
                {
                        f[i].need[j]=f[i].max[j]-f[i].alloc[j];
                        if(f[i].need[j]<0)
                                f[i].need[j]=0;
                }
}
count=0;
fl=0;
while(count!=n)
{
        g=0;
        for(j=0;j<n;j++)
        {
                if(f[j].flag==0)
                {
                        b=0;
                        for(p=0;p<r;p++)
                        {
                                if(avail[p]>=f[j].need[p])
                                        b=b+1;
                                else
                                        b=b-1;
                        }
                        if(b==r)
                        {
                                printf("\nP%d is executed",j);
                                seq[fl++]=j;
                                f[j].flag=1;
                                for(k=0;k<r;k++)
                                        avail[k]=avail[k]+f[j].alloc[k];
                                count=count+1;
                                printf("(");
                                for(k=0;k<r;k++)
                                        printf("%d\t ",avail[k]);
                                printf(")");
                                g=1;

                        }
                }
        }
        if(g==0)
        {
                printf("\nRequest not granted\t\tdeadlock occured");
                printf("\nSystem is in unsafe state");

        }
        printf("\nSytem is in safe state");
        printf("\nThe safe sequence is <");
        for(i=0;i<fl;i++)
                printf("P%d\t",seq[i]);
        printf(">");
```

```
        }
        getch();
}
```

Output:



C:\Users\Vishal\Desktop\B.exe

```
Enter no.of prcesses:
5

Enter no.of resources:3
Enter details of P0
Enter allocation:0 1 0

Enter max:7 5 3
Enter details of P1
Enter allocation:2 0 0

Enter max:3 2 2
Enter details of P2
Enter allocation:3 0 2

Enter max:9 0 2
Enter details of P3
Enter allocation:2 1 1

Enter max:2 2 2
Enter details of P4
Enter allocation:0 0 2

Enter max:4 3 3

Enter available resources:3 3 2

Enter new request details:
Enter process id:1

Enter request for resources:1 0 2

P1 is executed(5         3       2       )
P3 is executed(7         4       3       )
P4 is executed(7         4       5       )
Sytem is in safe state
The safe sequence is <P1        P3      P4      >
P0 is executed(7         5       5       )
P2 is executed(10        5       7       )
Sytem is in safe state
The safe sequence is <P1        P3      P4      P0      P2      >
```

Conclusion:

Hence, the banker’s algorithm is the deadlock avoidance algorithm. It is called so because it is used in banking systems to decide whether a loan can be granted or not. we successfully implement banker’s algorithm.