3) Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention

Aim:

Write a C program to simulate the Bankers Algorithm for Deadlock Avoidance.

Data structures

1. n- Number of process, m-number of resource types.

2. Available: Available[j]=k, k – instance of resource type Rj is available.

3. Max: If max [i, j]=k, Pi may request at most k instances resource Rj.

4. Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj

5. Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj,

6. Need [I, j] =Max [I, j]-Allocation [I, j];

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and

Finish[i] =False.

2. Find an i such that both

3. Finish[i] =False

4. Need<=Work

5. If no such I exist go to step 4.

6. work=work+Allocation, Finish[i] =True;

7. If Finish [1] =True for all I, then the system is in safe state.

Resource request algorithm

1. Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants

k instances of resource type Rj.

2. If Request<=Need I go to step 2. Otherwise raise an error condition.

3. If Request<=Available go to step 3. Otherwise Pi must since the resources are available.

4. Have the system pretend to have allocated the requested resources to process Pi by

modifying the state as follows;

5. Available=Available-Request I;

6. Allocation I =Allocation+Request I;

7. Need i=Need i-Request I;

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However, if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restore.

```
#include<stdio.h>
struct file
{
	int all[10];
	int max[10];
	int need[10];
	int flag;
};
void main()
{
	struct file f[10];
	int fl;
	int i,j,k,p,b,n,r,g,cnt=0,id,newr;
	int avail[10],seq[10];
	printf("Enter the no of processes--");
	scanf("%d",&n);
	printf("Enter the no of resources");
	scanf("%d",&r);
	for(i=0;i<n;i++)
	{
		printf("Enter the details for P%d\n",i);
		printf("Enter the allocation : ");
		for(j=0;j<r;j++)
			scanf("%d",&f[i].all[j]);
		printf("Enter the max: ");
```

```
        for(j=0;j<r;j++)
                scanf("%d",&f[i].max[j]);
        f[i].flag=0;
}
printf("\nEnter Available resources--\t");
for(i=0;i<r;i++)
        scanf("%d",&avail[i]);
printf("\nEnter new Request details--");
printf("\nEnter the pid\t--");
scanf("%d",&id);
printf("\nEnter the request for resources--\t");
for(i=0;i<r;i++)
{
        scanf("%d",&newr);
        f[id].all[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].all[j];
                if(f[i].need[j]<0)
                        f[i].need[j]=0;
        }
}
cnt=0;
fl=0;
while(cnt!=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 visited",j);
                                seq[fl++]=j;
                                f[j].flag=1;
                                for(k=0;k<r;k++)
                                        avail[k]=avail[k]+f[j].all[k];
                                cnt=cnt+1;
                                printf("(");
                                for(k=0;k<r;k++)
                                        printf("%3d",avail[k]);
                                printf(")");
                                g=1;
                        }
                }
```

```
                }

                if(g==0)
                {
                        printf("\nREQUEST NOT GRANTED__DEADLOCK OCCURRED");
                        printf("\nSYTEM IS IN UNSAFE STATE");
                        goto y;
                }
        }
        printf("\nSYSTEM IS IN SAFE STATE");
        printf("\nThe safe sequence is __(");
        for(i=0;i<fl;i++)
                printf("P%d\t",seq[i]);
        printf(")");
        y:printf("\nProcess\t\tAllocation\t   Max\t\t  Need\n");
        for(i=0;i<n;i++)
        {
                printf("P%d\t",i);
                for(j=0;j<r;j++)
                        printf("%6d",f[i].all[j]);
                for(j=0;j<r;j++)
                        printf("%6d",f[i].max[j]);
                for(j=0;j<r;j++)
                        printf("%6d",f[i].need[j]);
                printf("\n");
        }
}
```

INPUT:

Enter the no of processes--5
Enter the no of resources3
Enter the details for P0
Enter the allocation : 0 1 0
Enter the max: 7 5 3
Enter the details for P1
Enter the allocation : 2 0 0
Enter the max: 3 2 2
Enter the details for P2
Enter the allocation : 3 0 2
Enter the max: 9 0 2
Enter the details for P3
Enter the allocation : 2 1 1
Enter the max: 2 2 2
Enter the details for P4
Enter the allocation : 0 0 2
Enter the max: 4 3 3

Enter Available resources-- 3 3 2

Enter new Request details--
Enter the pid --1
Enter the request for resources-- 1 0 2

OUTPUT:

P1 is visited(5 3 2)
P3 is visited(7 4 3)
P4 is visited(7 4 5)
P0 is visited(7 5 5)
P2 is visited(10 5 7)
SYSTEM IS IN SAFE STATE

The safe sequence is _(P1 P3 P4 P0 P2)

Process	Allocation	Max	Need
P0	0 1 0	7 5 3	7 4 3
P1	3 0 2	3 2 2	0 2 0
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1