EX NO: 7A	**DEADLOCK AVOIDANCE – BANKER'S ALGORITHM**
DATE:	

AIM:

To implement deadlock avoidance by using Banker’s Algorithm.

Banker’s Algorithm:

When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

Data structures

- n-Number of process, m-number of resource types.
- Available: Available[j]=k, k – instance of resource type Rj is available.
- Max: If max[i, j]=k, Pi may request at most k instances resource Rj.
- Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj
- Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj, 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 Finish[i] =False
    Need<=Work
   If no such I exists go to 4.
3. work=work+Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state.

Resource request algorithm

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.

1. if Request<=Need I go to 2. Otherwise raise an error condition.
2. if Request<=Available go to 3. Otherwise Pi must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows; Available=Available-Request I;
   Allocation I =Allocation+Request I;
   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 restored.

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. Or not we allow the request.
10. Stop the program.

PROGRAM:

/* BANKER'S ALGORITHM */

```
#include<stdio.h>

struct da
{
   int max[10],a1[10],need[10],before[10],after[10];
}p[10];
void
main()
{        int i,j,k,l,r,n,tot[10],av[10],cn=0,cz=0,temp=0,c=0;
 clrscr();
 printf("\n ENTER THE NO. OF PROCESSES:");  scanf("%d",&n);
 printf("\n ENTER THE NO. OF RESOURCES:");
  scanf("%d",&r);
 for(i=0;i<n;i++)
 {
 printf("PROCESS %d \n",i+1);
   for(j=0;j<r;j++)
   {
 printf("MAXIMUM VALUE FOR RESOURCE %d:",j+1);
 scanf("%d",&p[i].max[j]);
   }
  for(j=0;j<r;j++)
   {
 printf("ALLOCATED FROM RESOURCE %d:",j+1);
  scanf("%d",&p[i].a1[j]);        p[i].need[j]=p[i].max[j]-p[i].a1[j];
   }
 }
 for(i=0;i<r;i++)
 {
          printf("ENTER TOTAL VALUE OF RESOURCE %d:",i+1);
 scanf("%d",&tot[i]);
 }
 for(i=0;i<r;i++)
 {
          for(j=0;j<n;j++)
 temp=temp+p[j].a1[i];
 av[i]=tot[i]-temp;                          temp=0;
 }
```

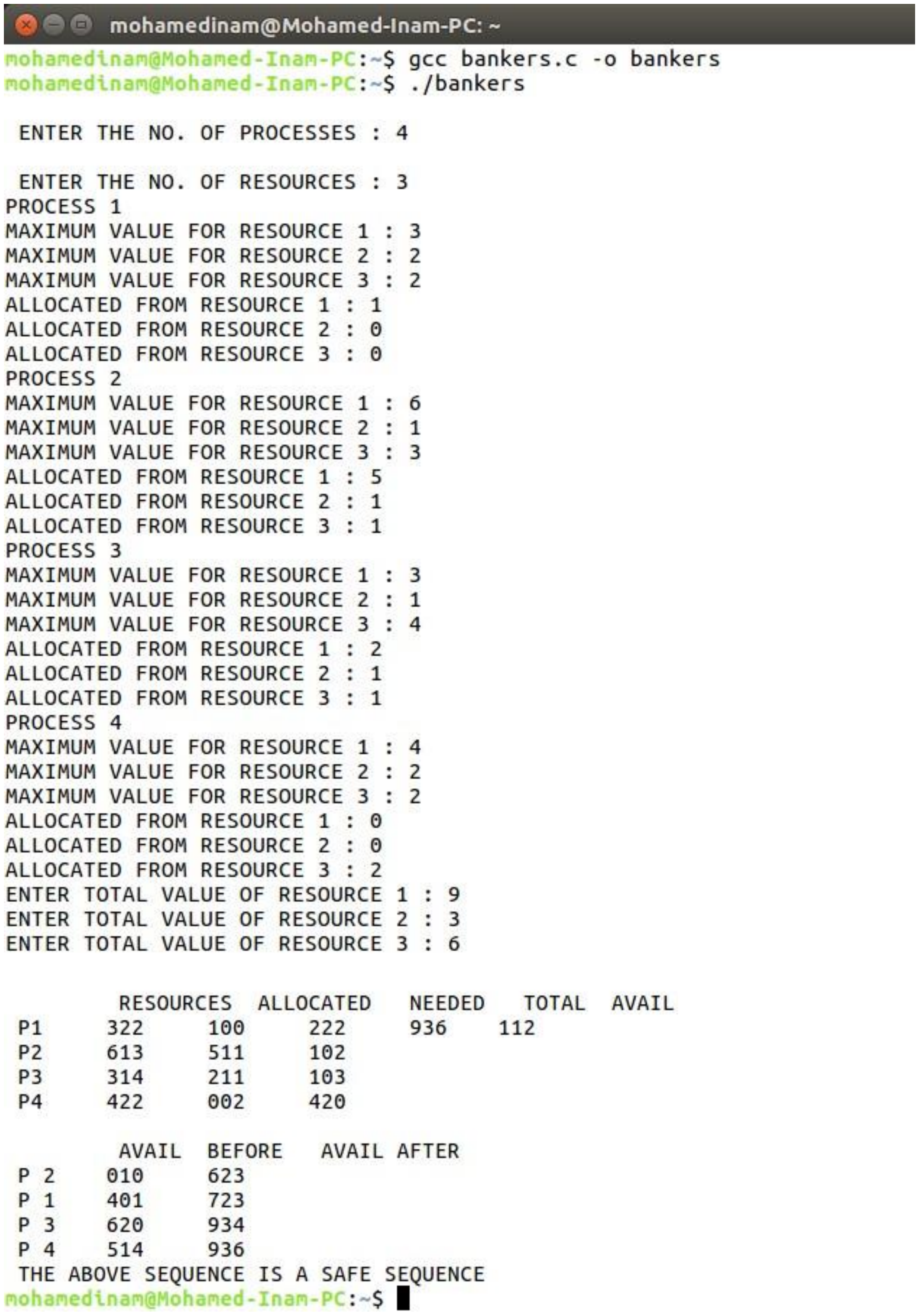
```
printf("\n\t RESOURCES  ALLOCATED   NEEDED   TOTAL  AVAIL");
for(i=0;i<n;i++)
  {
 printf("\n P%d \t",i+1);
    for(j=0;j<r;j++)
  printf("%d",p[i].max[j]);
  printf("\t");
    for(j=0;j<r;j++)
 printf("%d",p[i].a1[j]);
 printf("\t");
    for(j=0;j<r;j++)
 printf("%d",p[i].need[j]);
 printf("\t");
    for(j=0;j<r;j++)
    {
   if(i==0)
   printf("%d",tot[j]);
    }
printf("
");
    for(j=0;j<r;j++)
   {        if(i==0)
printf("%d",av[j]);
   }
 }
printf("\n\n\t AVAIL  BEFORE\T AVAIL AFTER ");
for(l=0;l<n;l++)
   {
for(i=0;i<n;i++)
    {
  for(j=0;j<r;j++)
  {
    if(p[i].need[j] >av[j])
cn++;
    if(p[i].max[j]==0)       cz++;
  }
if(cn==0 && cz!=r)
    {
      for(j=0;j<r;j++)
      {
 p[i].before[j]=av[j]-p[i].need[j];
 p[i].after[j]=p[i].before[j]+p[i].max[j];          av[j]=p[i].after[j];
 p[i].max[j]=0;
```

```
      }
printf("\n P %d \t",i+1);
       for(j=0;j<r;j++)
             printf("%d",p[i].before[j]);    printf("\t");
       for(j=0;j<r;j++)
             printf("%d",p[i].after[j]);
             cn=0;
cz=0;    c++;    break;
   }    else
   {
     cn=0;cz=0;
   }
  } }  if(c==n)
 printf("\n THE ABOVE SEQUENCE IS A SAFE SEQUENCE");     else
            printf("\n DEADLOCK OCCURED");
```

```
}
```

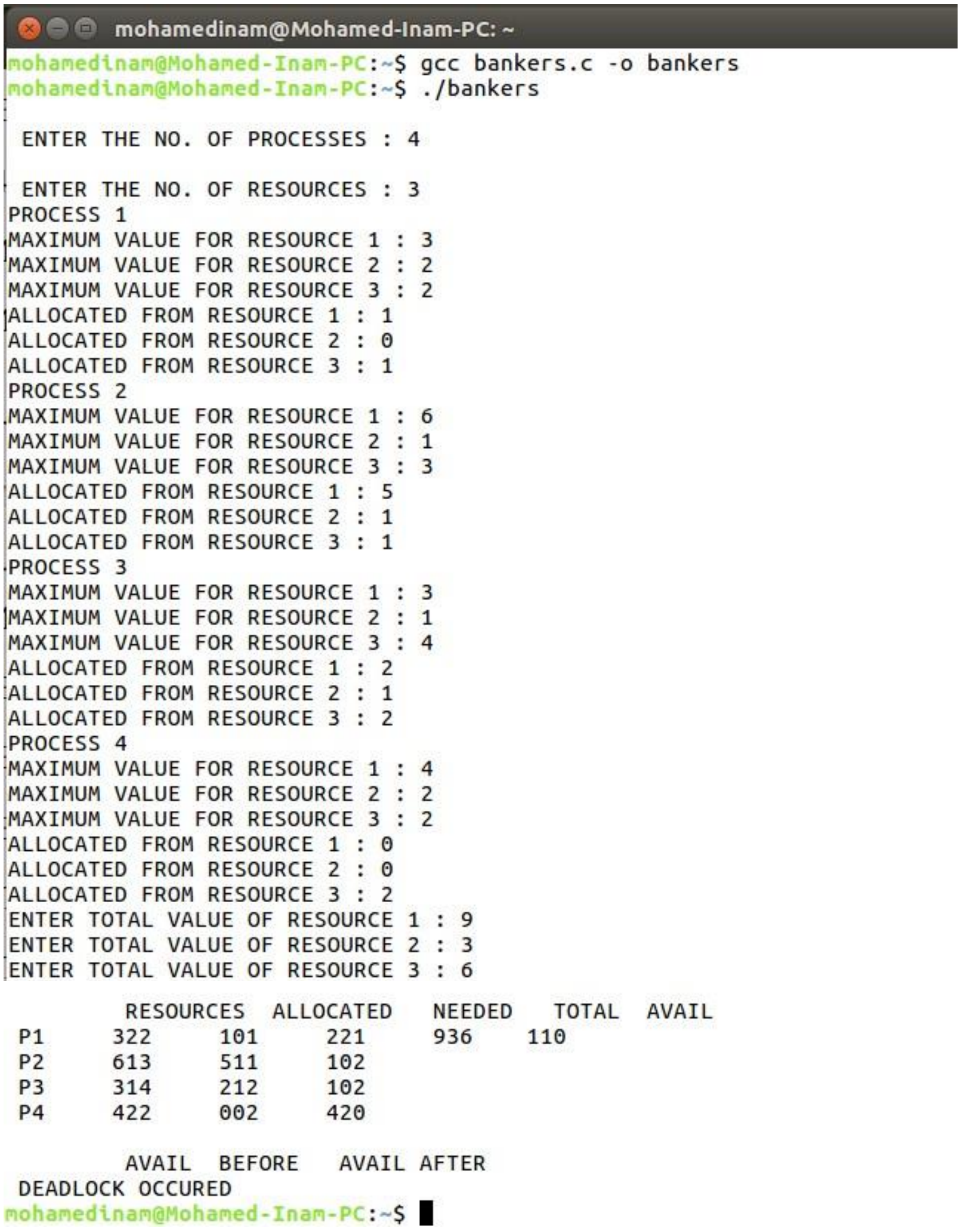
OUTPUT:

//RUN: NO deadlock



```
mohamedinam@Mohamed-Inam-PC: ~
mohamedinam@Mohamed-Inam-PC:~$ gcc bankers.c -o bankers
mohamedinam@Mohamed-Inam-PC:~$ ./bankers

 ENTER THE NO. OF PROCESSES : 4

 ENTER THE NO. OF RESOURCES : 3
PROCESS 1
MAXIMUM VALUE FOR RESOURCE 1 : 3
MAXIMUM VALUE FOR RESOURCE 2 : 2
MAXIMUM VALUE FOR RESOURCE 3 : 2
ALLOCATED FROM RESOURCE 1 : 1
ALLOCATED FROM RESOURCE 2 : 0
ALLOCATED FROM RESOURCE 3 : 0
PROCESS 2
MAXIMUM VALUE FOR RESOURCE 1 : 6
MAXIMUM VALUE FOR RESOURCE 2 : 1
MAXIMUM VALUE FOR RESOURCE 3 : 3
ALLOCATED FROM RESOURCE 1 : 5
ALLOCATED FROM RESOURCE 2 : 1
ALLOCATED FROM RESOURCE 3 : 1
PROCESS 3
MAXIMUM VALUE FOR RESOURCE 1 : 3
MAXIMUM VALUE FOR RESOURCE 2 : 1
MAXIMUM VALUE FOR RESOURCE 3 : 4
ALLOCATED FROM RESOURCE 1 : 2
ALLOCATED FROM RESOURCE 2 : 1
ALLOCATED FROM RESOURCE 3 : 1
PROCESS 4
MAXIMUM VALUE FOR RESOURCE 1 : 4
MAXIMUM VALUE FOR RESOURCE 2 : 2
MAXIMUM VALUE FOR RESOURCE 3 : 2
ALLOCATED FROM RESOURCE 1 : 0
ALLOCATED FROM RESOURCE 2 : 0
ALLOCATED FROM RESOURCE 3 : 2
ENTER TOTAL VALUE OF RESOURCE 1 : 9
ENTER TOTAL VALUE OF RESOURCE 2 : 3
ENTER TOTAL VALUE OF RESOURCE 3 : 6

         RESOURCES  ALLOCATED   NEEDED   TOTAL  AVAIL
 P1     322       100       222       936    112
 P2     613       511       102
 P3     314       211       103
 P4     422       002       420

         AVAIL  BEFORE   AVAIL AFTER
 P 2    010      623
 P 1    401      723
 P 3    620      934
 P 4    514      936
 THE ABOVE SEQUENCE IS A SAFE SEQUENCE
mohamedinam@Mohamed-Inam-PC:~$
```

//RUN2: Deadlock occurs



```
mohamedinam@Mohamed-Inam-PC: ~
mohamedinam@Mohamed-Inam-PC:~$ gcc bankers.c -o bankers
mohamedinam@Mohamed-Inam-PC:~$ ./bankers

 ENTER THE NO. OF PROCESSES : 4

 ENTER THE NO. OF RESOURCES : 3
PROCESS 1
MAXIMUM VALUE FOR RESOURCE 1 : 3
MAXIMUM VALUE FOR RESOURCE 2 : 2
MAXIMUM VALUE FOR RESOURCE 3 : 2
ALLOCATED FROM RESOURCE 1 : 1
ALLOCATED FROM RESOURCE 2 : 0
ALLOCATED FROM RESOURCE 3 : 1
PROCESS 2
MAXIMUM VALUE FOR RESOURCE 1 : 6
MAXIMUM VALUE FOR RESOURCE 2 : 1
MAXIMUM VALUE FOR RESOURCE 3 : 3
ALLOCATED FROM RESOURCE 1 : 5
ALLOCATED FROM RESOURCE 2 : 1
ALLOCATED FROM RESOURCE 3 : 1
PROCESS 3
MAXIMUM VALUE FOR RESOURCE 1 : 3
MAXIMUM VALUE FOR RESOURCE 2 : 1
MAXIMUM VALUE FOR RESOURCE 3 : 4
ALLOCATED FROM RESOURCE 1 : 2
ALLOCATED FROM RESOURCE 2 : 1
ALLOCATED FROM RESOURCE 3 : 2
PROCESS 4
MAXIMUM VALUE FOR RESOURCE 1 : 4
MAXIMUM VALUE FOR RESOURCE 2 : 2
MAXIMUM VALUE FOR RESOURCE 3 : 2
ALLOCATED FROM RESOURCE 1 : 0
ALLOCATED FROM RESOURCE 2 : 0
ALLOCATED FROM RESOURCE 3 : 2
ENTER TOTAL VALUE OF RESOURCE 1 : 9
ENTER TOTAL VALUE OF RESOURCE 2 : 3
ENTER TOTAL VALUE OF RESOURCE 3 : 6

         RESOURCES  ALLOCATED   NEEDED   TOTAL  AVAIL
 P1     322      101      221      936     110
 P2     613      511      102
 P3     314      212      102
 P4     422      002      420

         AVAIL  BEFORE   AVAIL AFTER
 DEADLOCK OCCURED
mohamedinam@Mohamed-Inam-PC:~$
```

RESULT

Thus the banker’s algorithm is implemented successfully for Deadlock avoidance & Dead Lock Prevention.