Ex.No:8 BANKERS ALGORITHM FOR DEAD LOCK AVOIDANCE

Date:

AIM

To implement deadlock avoidance by using Banker's Algorithm.

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 or not if we allow the request.
9. Stop.

PROGRAM

```
#include <stdio.h>
#include <stdio.h>

main()
{
   int r[1][10], av[1][10];
   int all[10][10], max[10][10], ne[10][10], w[10],safe[10];
   int i=0, j=0, k=0, l=0, np=0, nr=0, count=0, cnt=0;

   clrscr();
   printf("enter the number of processes in a system");
   scanf("%d", &np);
   printf("enter the number of resources in a system");
   scanf("%d",&nr);
   for(i=1; i<=nr; i++)
   {
      printf("Enter no. of instances of resource R%d " ,i);
      scanf("%d", &r[0][i]);
      av[0][i] = r[0][i];
   }

   for(i=1; i<=np; i++)
      for(j=1; j<=nr; j++)
         all[i][j] = ne[i][j] = max[i][j] = w[i]=0;
```

```
printf("Enter the allocation matrix");
for(i=1; i<=np; i++)
{
   for(j=1; j<=nr; j++)
   {
       scanf("%d", &all[i][j]);
       av[0][j] = av[0][j] - all[i][j];
   }
}

printf("Enter the maximum matrix");
for(i=1; i<=np; i++)
{
   for(j=1; j<=nr; j++)
   {
         scanf("%d",&max[i][j]);
   }
}

for(i=1; i<=np; i++)
{
   for(j=1; j<=nr; j++)
   {
      ne[i][j] = max[i][j] - all[i][j];
   }
}

for(i=1; i<=np; i++)
{
   printf("pocess P%d", i);
   for(j=1; j<=nr; j++)
   {
      printf("\n allocated %d\t",all[i][j]);
      printf("maximum %d\t",max[i][j]);
      printf("need %d\t",ne[i][j]);
   }
   printf("\n_________________________\n");
}

printf("\nAvailability ");
for(i=1; i<=nr; i++)
   printf("R%d %d\t", i, av[0][i]);
printf("\n____________");
printf("\n safe sequence");
```

```
   for(count=1; count<=np; count++)
   {
     for(i=1; i<=np; i++)
     {
         Cnt = 0;
         for(j=1; j<=nr; j++)
         {
            if(ne[i][j] <= av[0][j] && w[i]==0)
               cnt++;
         }
         if(cnt == nr)
         {
            k++;
            safe[k] = i;
            for(l=1; l<=nr; l++)
               av[0][l] = av[0][l] + all[i][l];
            printf("\n P%d ",safe[k]);
            printf("\t Availability ");
            for(l=1; l<=nr; l++)
               printf("R%d %d\t", l, av[0][l]);
            w[i]=1;
         }
      }
   }
   getch();
}
```

Output

```
enter the number of processes in a system 3
enter the number of resources in a system 3

enter no. of instances of resource R1 10
enter no. of instances of resource R2 7
enter no. of instances of resource R3 7

Enter the allocation matrix
3 2 1
1 1 2
4 1 2

Enter the maximum matrix
4 4 4
3 4 5
5 2 4

pocess P1
 allocated 3    maximum 4       need 1
 allocated 2    maximum 4       need 2
 allocated 1    maximum 4       need 3
```

```
_________________________
pocess P2
 allocated 1    maximum 3       need 2
 allocated 1    maximum 4       need 3
 allocated 2    maximum 5       need 3
_________________________
pocess P3
 allocated 4    maximum 5       need 1
 allocated 1    maximum 2       need 1
 allocated 2    maximum 4       need 2
_________________________

Availability R1 2        R2 3    R3 2
____________
 safe sequence
 P3      Availability R1 6       R2 4    R3 4
 P1      Availability R1 9       R2 6    R3 5
 P2      Availability R1 10      R2 7    R3 7
```

Result

Thus bankers algorithm for dead lock avoidance was executed successfully.