Ex. No.: 9
Date: 29|3|25

DEADLOCK AVOIDANCE

Aim:

To find out a safe sequence using Banker's algorithm for deadlock avoidance.

Algorithm:

```
1. Initialize work=available and finish[i]=false for all values of i
2. Find an i such that both:
   finish[i]=false and Needᵢ<= work
3. If no such i exists go to step 6
4. Compute work=work+allocationi
5. Assign finish[i] to true and go to step 2
6. If finish[i]==true for all i, then print safe sequence
7. Else print there is no safe sequence
```

Program Code:

```
#Include <stdio.h>.
# Include <stdbook.h>
# define P4
# define P3.
bool is safe (int process[] , int available [], int max [][R],
                   int allocation [] [P])
{
    int need [P] [R].;
    int work [R]:
    bool fin [P] = {false} :
    int safe sequence [P];
    for (int i=0; i<p ; i++)
    for (int ==0; j<p ;j++)
              need [i] [j] = max [i] [j] - allocation
                                              [i] [j] .

 for (int i=0; i<R ; i++)
              work [i] - available [i];

int c=0;
while (c<P) {
```

```
bool found = false;
for (int i=0; i<p; i++){
        if (!fin [i]){
            bool canAllocate = true;
            for (int j=0; j<R; j++)
               if (need [i] [j] > work [j])
               {
                   canAllocate = false;
                   break;
               }
               if (canAllocate){
                  for (int j=0; j<R; j++)
                     work[j] += Allocation [i][j];

               safeSequence [c++] = process [i].
               fin [i] = true;
               found = true;
            }
        }
    }
    if (!found){
        printf ("No safe sequence\n");
        return false;
    }
}
printf ("The need matrix is \n");
for (int i=0; i<p; i++)
{
    for (int j=0; j<R; j++)
    {
        printf (" %d\t", need [i][j]);
    }
    printf ("\n");
}
printf ("The safe sequence is \n");
```

```
for (int i=0; i<p; i++){
        printf ("P %d ", safesequence[i]);
}   if (i != p-1){
            print ("->");
}
  printf("\n");
 return , true ;
}
int main()
{
    int  process [] = {0,1,2,3};
    int available [] = {2,2,3};
    int max [P] [R]    = { {4,3,3}, {8,0,1} . {3,2,2}
                                   {1,0,2,3}.
   int allocation [P] [R] =  {{0,1,0}, {2,0,0}, {3,0,2}
                                        {1,0,1}}
 is Safe (process, available, max, allocation);
 return 0;
}
```

Output

The need matrix

```
4   2   3
6   0   1
0   2   0
0   0   1
```

The safe sequence is

P2 → P3 → P0 → P1

Sample Output:

```
The SAFE Sequence is
P1 -> P3 -> P4 -> P0 -> P2
```

Result:

Thus the above program to find out a safe. sequence using Bankers Algorithm for deadlock avoidance has been executed successfully.