Program:

Aim :Write a C program to simulate Bankers Algorithm for Deadlock Prevention

```
#include<stdio.h>
#include<unistd.h>
int main()
{
   int max[10][10],alloc[10][10],need[10][10],avail[10],i,j,p,r,finish[10]={0},flag=0;
    printf("\n SIMULATION OF DEADLOCK PREVENTION \n ");
    printf("Enter no. of processes, resources\n ");
    scanf("%d%d",&p,&r);
    printf("Enter allocation matrix");
    for(i=0;i<p;i++)
    {
       for(j=0;j<r;j++)
       {
          scanf("%d",&alloc[i][j]);
          }
    }
    printf("\n enter max matrix");
    for(i=0;i<p;i++)
    {
       for(j=0;j<r;j++)
       {
          scanf("%d",&max[i][j]);
       }
     }
    printf(" \n enter available matrix");
    for(i=0;i<r;i++)
    {
       scanf("%d",&avail[i]);
     }
   for(i=0;i<p;i++)
   {
       for(j=0;j<r;j++)
       {
          need[i][j]=max[i][j]-alloc[i][j];
          //func(); /*calling function*/
           if(flag==0)
           {
              if(finish[i]!=1)
              {
               printf("\n Failing :Mutual exclusion");
              for(j=0;j<r;j++)
              { /*checking for mutual exclusion*/
                   if(avail[j]<need[i][j])
                   avail[j]=need[i][j];
               }
              // func();
              printf("\n By allocating required resources to process %d dead lock is prevented ",i);
             printf("\n lack of preemption");
             }
            }
```

```
            }
          }
      for(j=0;j<r;j++)
      {
           if(avail[j]<need[i][j])
           avail[j]=need[i][j];
          alloc[i][j]=0;
      }
    // func();
     printf("\n dead lock is prevented by allocating needed resources");
      printf(" \n failing:Hold and Wait condition ");
       for(j=0;j<r;j++)
       { /*checking hold and wait condition*/
            if(avail[j]<need[i][j])
             avail[j]=need[i][j];
        }
         //func( );
         printf("\n AVOIDING ANY ONE OF THE CONDITION, U CAN PREVENT DEADLOCK");
        //func();
        while(1)
        {
           for(flag=0,i=0;i<p;i++)
           {
                 if(finish[i]==0)
                  {
                       for(j=0;j<r;j++)
                       {
                           if(need[i][j]<=avail[j])
                           continue;
                       else
                       break;
                       }
                      if(j==r)
                       {
                          for(j=0;j<r;j++)
                          avail[j]+=alloc[i][j];
                          flag=1;
                          finish[i]=1;
                         }
                    }
             }
      }
      return 0;

}
```

Output:

```
 SIMULATION OF DEADLOCK PREVENTION
 Enter no. of processes, resources
 5 3
Enter allocation matrix1 2 0 1
1 1 0
3 0 2
0 4 1
2 1 1

 enter max matrix 2 1 1
3 2 1
3 1 3
1 2 1
4 1 2

 enter available matrix1 2 3

 Failing :Mutual exclusion
 By allocating required resources to process 0 dead lock is prevented
 lack of preemption
 Failing :Mutual exclusion
 By allocating required resources to process 1 dead lock is prevented
 lack of preemption
 Failing :Mutual exclusion
 By allocating required resources to process 2 dead lock is prevented
 lack of preemption
 Failing :Mutual exclusion
 By allocating required resources to process 3 dead lock is prevented
 lack of preemption
 Failing :Mutual exclusion
 By allocating required resources to process 4 dead lock is prevented
 lack of preemption
 dead lock is prevented by allocating needed resources
 failing:Hold and Wait condition
```