Lab-05

**Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance**.

#include <stdio.h>

int main()

{

int n, m, i, j, k;

printf("Enter the number of processes: ");

scanf("%d", &n);

printf("Enter the number of resources: ");

scanf("%d", &m);

int allocation[n][m];

printf("Enter the Allocation Matrix:\n");

for (i = 0; i < n; i++)

{

for (j = 0; j < m; j++)

{

scanf("%d", &allocation[i][j]);

}

}

int max[n][m];

printf("Enter the MAX Matrix:\n");

for (i = 0; i < n; i++)

{

for (j = 0; j < m; j++)

{

scanf("%d", &max[i][j]);

}

}

int available[m];

printf("Enter the Available Resources:\n");

for (i = 0; i < m; i++)

{

scanf("%d", &available[i]);

}

int f[n], ans[n], ind = 0;

for (k = 0; k < n; k++)

{

f[k] = 0;

}

int need[n][m];

for (i = 0; i < n; i++)

{

for (j = 0; j < m; j++)

{

need[i][j] = max[i][j] - allocation[i][j];

}

}

int safe = 1;

for (k = 0; k < n; k++)

{

int flag = 0;

for (i = 0; i < n; i++)

{

if (f[i] == 0)

{

int can\_finish = 1;

// Check if all the needs of the process are met

for (j = 0; j < m; j++)

{

if (need[i][j] > available[j])

{

can\_finish = 0;

break;

}

}

if (can\_finish)

{

ans[ind++] = i; // Add to safe sequence

for (int y = 0; y < m; y++)

{

available[y] += allocation[i][y];

}

f[i] = 1;

flag = 1;

break;

}

}

}

if (!flag)

{

safe = 0;

break;

}

}

if (safe)

{

printf("The system is in a safe state.\n");

printf("The safe sequence is: ");

for (i = 0; i < n; i++)

{

printf("P%d ", ans[i]);

}

printf("\n");

}

else

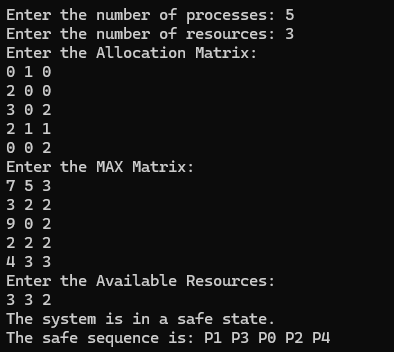
{

printf("The system is not in a safe state.\n");

}

return 0;

}



**Write a C program to simulate deadlock detection.**

#include<stdio.h>  
static int mark[20];  
int i,j,np,nr;  
  
int main()  
{  
int alloc[10][10],request[10][10],avail[10],r[10],w[10];  
  
printf("\nEnter the no of process: ");  
scanf("%d",&np);  
printf("\nEnter the no of resources: ");  
scanf("%d",&nr);  
for(i=0;i<nr;i++)  
{  
printf("\nTotal Amount of the Resource R%d: ",i+1);  
scanf("%d",&r[i]);  
}  
  
  
  
  
printf("\nEnter the request matrix:");  
  
for(i=0;i<np;i++)  
for(j=0;j<nr;j++)  
scanf("%d",&request[i][j]);  
  
printf("\nEnter the allocation matrix:");  
for(i=0;i<np;i++)  
for(j=0;j<nr;j++)  
scanf("%d",&alloc[i][j]);  
/\*Available Resource calculation\*/  
for(j=0;j<nr;j++)  
{  
avail[j]=r[j];  
for(i=0;i<np;i++)  
{  
avail[j]-=alloc[i][j];  
  
}  
}  
  
//marking processes with zero allocation  
  
for(i=0;i<np;i++)  
{  
int count=0;  
 for(j=0;j<nr;j++)  
   {  
      if(alloc[i][j]==0)  
        count++;  
      else  
        break;  
    }  
 if(count==nr)  
 mark[i]=1;  
}  
// initialize W with avail  
  
for(j=0;j<nr;j++)  
    w[j]=avail[j];  
  
//mark processes with request less than or equal to W  
for(i=0;i<np;i++)  
{  
int canbeprocessed=0;  
 if(mark[i]!=1)  
{  
   for(j=0;j<nr;j++)  
    {  
      if(request[i][j]<=w[j])  
        canbeprocessed=1;  
      else  
         {  
         canbeprocessed=0;  
         break;  
          }  
     }  
if(canbeprocessed)  
{  
mark[i]=1;  
  
for(j=0;j<nr;j++)  
w[j]+=alloc[i][j];  
}  
}  
}  
  
//checking for unmarked processes  
int deadlock=0;  
for(i=0;i<np;i++)  
if(mark[i]!=1)  
deadlock=1;  
  
  
if(deadlock)  
printf("\n Deadlock detected");  
else  
printf("\n No Deadlock possible");  
}

