**Lab 13-Dead lock Detection Algorithm**

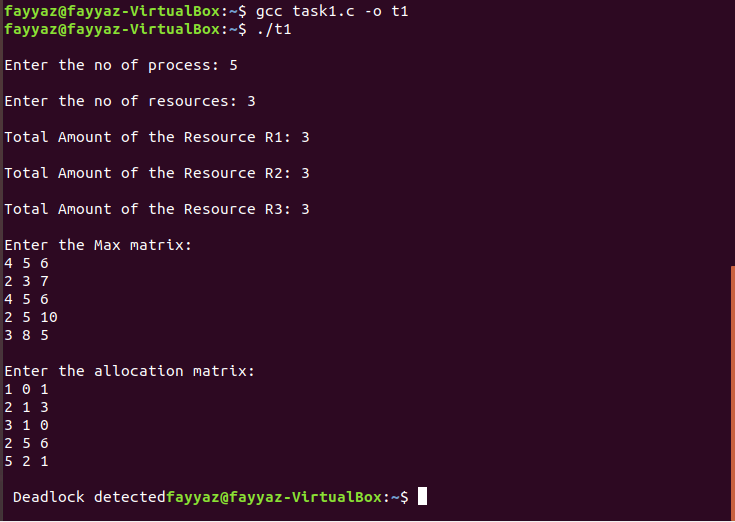
**Tasks**

1. **Implement the above algorithm using C programming language to detect the deadlock in the operating systems.**

**Code:**

#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");  
}

**Output:**



1. **Design a C program and implement the recovery of the detected deadlock in the operating system as in the above program. You can take help from the above algorithm.**

**Code:**

#include <stdio.h>  
int main()  
{  
int max[5][3], alloc[5][3], avail[3];  
int n, m, i, j, k;  
printf("Enter the no of processes : ");  
scanf("%d", &n);  
printf("\nEnter the no of resources : ");  
scanf("%d", &m);  
printf("\nEnter the Max Matrix for each process : ");  
for(int i = 0; i < n; i++)  
{  
printf("\nFor process %d : ", i + 1);  
for(int j = 0; j < m; j++)  
{  
scanf("%d", &max[i][j]);  
}  
  
}  
printf("\n\nEnter the allocation for each process : ");  
for(i = 0; i < n; i++)  
{  
printf("\nFor process %d : ",i + 1);  
for(j = 0; j < m; j++)  
{  
scanf("%d", &alloc[i][j]);  
}  
}  
printf("\n\nEnter the Available Resources : ");  
for(i = 0; i < m; i++)  
{  
scanf("%d", &avail[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] - alloc[i][j];  
}  
int y = 0;  
for (k = 0; k < 5; k++) {  
for (i = 0; i < n; i++) {  
if (f[i] == 0) {  
int flag = 0;  
for (j = 0; j < m; j++) {  
if (need[i][j] > avail[j]){  
flag = 1;  
break;  
}  
}  
if (flag == 0) {  
ans[ind++] = i;  
for (y = 0; y < m; y++)  
avail[y] += alloc[i][y];  
f[i] = 1;  
}  
}  
}  
}  
printf("Following is the SAFE Sequence\n");  
for (i = 0; i < n - 1; i++)  
printf(" P%d ->", ans[i]);  
printf(" P%d", ans[n - 1]);  
return (0);  
  
}

**Output:**

