**Mini Project**

**Title:** Write a program for deadlock detection.

**Description:**

If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur.

In this environment, the system may provide:

* An algorithm that examines the state of the system to determine whether a deadlock has occurred
* An algorithm to recover from the deadlock

Single Instance of Each Resource Type

If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of the resource-allocation graph, called a wait-for graph.

We obtain this graph from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges.

More precisely, an edge from Pi to Pj in a wait-for graph implies that process Pi is waiting for process Pj to release a resource that Pi needs.

An edge Pi → Pj exists in a wait-for graph if and only if the corresponding resource allocation graph contains two edges Pi → Rq and Rq → Pj for some resource Rq.

As before, a deadlock exists in the system if and only if the wait-for graph contains a cycle.

To detect deadlocks, the system needs to maintain the wait for graph and periodically invoke an algorithm that searches for a cycle in the graph.

Several Instances of a Resource Type

The wait-for graph scheme is not applicable to a resource-allocation system with multiple instances of each resource type.

We turn now to a deadlock detection algorithm that is applicable to such a system.

The algorithm employs several time-varying data structures:

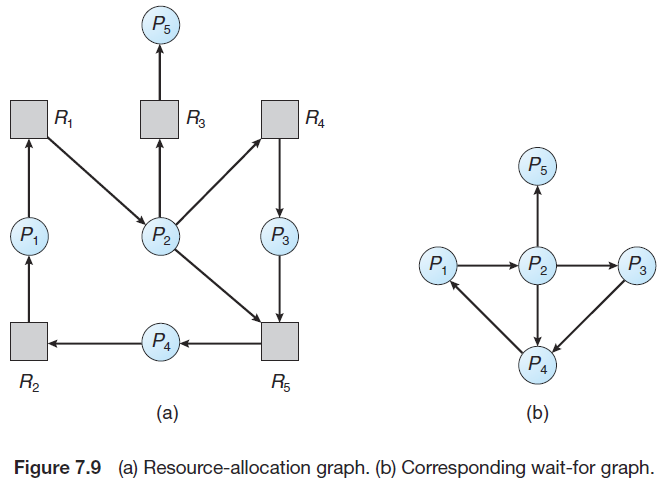
• Available: A vector of length m indicates the number of available resources of each type.

• Allocation. An n × m matrix defines the number of resources of each type currently allocated to each process.

• Request. An n × m matrix indicates the current request of each process.

To simplify notation, we again treat the rows in the matrices Allocation and Request as vectors; we refer to them as Allocation i and Request i.

The detection algorithm described here investigates every possible allocation sequence for the processes that remain to be completed.



Algorithm:

1. Let Work and Finish be vectors of length m and n, respectively.

Initialize Work = Available. For i = 0, 1... N–1, if Allocation i not equal to 0, then Finish[i] = false. Otherwise, Finish[i] = true.

2. Find an index i such that both

a. Finish[i] == false

b. Request i ≤Work

If no such i exists, go to step 4.

3. Work =Work + Allocation i

Finish[i] = true

Go to step 2.

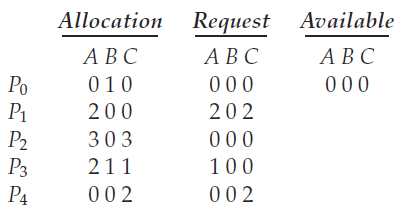
4. If Finish[i] ==false for some i, 0≤i<n, then the system is in a deadlocked state. Moreover, if Finish[i] == false, then process Pi is deadlocked.

Example

Consider a system with five processes P0 through P4 and three resource types A, B, and C.

Resource type A has seven instances, resource type B has two instances, and resource type C has six instances.

Suppose that, at time T0, we have the following resource-allocationstate:

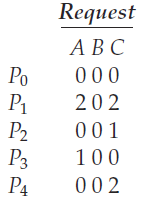


We claim that the system is not in a deadlocked state.

Indeed, if we execute our algorithm, we will find that the sequence <P0, P2, P3, P1, P4> results in Finish[i] == true for all i.

Suppose now that process P2 makes one additional request for an instance of type C.

The Request matrix is modified as follows:



We claim that the system is now deadlocked. Although we can reclaim the resources held by process P0, the number of available resources is not sufficient to fulfill the requests of the other processes.

Thus, a deadlock exists, consisting of processes P1, P2, P3, and P4.

**Program Code:**

#include<stdio.h>

int max[100][100];

int alloc[100][100];

int avail[100];

int n,r;

void input();

void display();

void calculate();

int main()

{

printf("\nDEADLOCK DETECTION ALGORITHM\n");

input();

display();

calculate();

return 0;

}

void input()

{

int i,j;

printf("\nEnter the no of Processes: ");

scanf("%d",&n);

printf("Enter the no of resource instances: ");

scanf("%d",&r);

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

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

for(j=0;j<r;j++) {

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

}

}

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

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

{

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

{

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

}

}

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

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

{

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

}

}

void display()

{

int i,j;

printf("Process\t\tMax\t\tAllocation\t\tAvailable");

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

{

printf("\nP%d\t\t\t",i+1);

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

{

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

}

printf("\t\t");

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

{

printf("%d ",alloc[i][j]);

}

printf("\t\t");

if(i==0)

{

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

{

printf("%d ",avail[j]);

}

}

}

}

void calculate()

{

int finish[100],dead[100],need[100][100];

int flag=1,i,j,k;

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

{

finish[i]=0;

}

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

{

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

{

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

}

}

while(flag==1)

{

flag=0;

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

{

int c=0;

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

{

if((finish[i]==0)&&(need[i][j]<=avail[j]))

{

c++;

if(c==r)

{

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

{

avail[k]= avail[k] + alloc[i][j];

finish[i]=1;

flag=1;

}

if(finish[i]==1)

{

i=n;

}

}

}

}

}

}

j=0;

flag=0;

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

{

if(finish[i]==0)

{

dead[j]=i;

j++;

flag=1;

}

}

if(flag==1)

{

printf("\n\nSystem is in Deadlock and the Deadlock process are:\n");

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

{

printf("P%d\t",dead[i]);

}

}

else

{

printf("\n\nNo Deadlock Detected");

}

}

**Output:**

