Program to Simulate bankers algorithm for DeadLock Avoidance

BY • OCTOBER 21, 2019

Concept: Deadlock is a situation where in two or more competing actions are waiting f or the other to finish, and thus neither ever does. When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether them allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

Data structures

n-Number of process, m-number of resource types.

Available: Available[j]=k, k – instance of resource type Rj is available.

Max: If max[i, j]=k, Pi may request at most k instances resource Rj.

Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj

Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj, Need[I, j]=Max[I, j]-Allocation[I, j];

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.

2. Find an i such that both Finish[i] =False Need<=Work If no such I exists go to step 4.

3. work= work + Allocation, Finish[i] =True;

4. if Finish[1]=True for all I, then the system is in safe state. Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2. Otherwise raise an error condition.

2. if Request<=Available go to step 3. Otherwise Pi must since the resources are available.

3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;

Available=Available-Request I;

Allocation I =Allocation +Request I;

Need i=Need i-Request I;

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

ALGORITHM:

1. Start .

2. Get the values of resources and processes.

3. Get the avail value.

4. After allocation find the need value.

5. Check whether its possible to allocate.

6. If it is possible then the system is in safe state.

7. Else system is not in safety state.

8. If the new request comes then check that the system is in safety.

9. or not if we allow the request.

10. stop .

#include

#include

int max[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n,r;

void input();

void show();

void cal();

int main()

{

int i,j;

printf("\*\*\*\*\*\*\*\*\*\* Banker's Algo \*\*\*\*\*\*\*\*\*\*\*\*\n");

input();

show();

cal();

getch();

return 0;

}

void input()

{

int i,j;

printf("Enter the no of Processes\t");

scanf("%d",&n);

printf("Enter the no of resources instances\t");

scanf("%d",&r);

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

for(i=0;i<n;i++)< span="" style="box-sizing: border-box;">

{

for(j=0;j<r;j++)< span="" style="box-sizing: border-box;">

{

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

}

}

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

for(i=0;i<n;i++)< span="" style="box-sizing: border-box;">

{

for(j=0;j<r;j++)< span="" style="box-sizing: border-box;">

{

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

}

}

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

for(j=0;j<r;j++)< span="" style="box-sizing: border-box;">

{

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

}

}

void show()

{

int i,j;

printf("Process\t Allocation\t Max\t Available\t");

for(i=0;i<n;i++)< span="" style="box-sizing: border-box;">

{

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

for(j=0;j<r;j++)< span="" style="box-sizing: border-box;">

{

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

}

printf("\t");

for(j=0;j<r;j++)< span="" style="box-sizing: border-box;">

{

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

}

printf("\t");

if(i==0)

{

for(j=0;j<r;j++)< span="" style="box-sizing: border-box;">

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

}

}

}

void cal()

{

int finish[100],temp,need[100][100],flag=1,k,c1=0;

int safe[100];

int i,j;

for(i=0;i<n;i++)< span="" style="box-sizing: border-box;">

{

finish[i]=0;

}

//find need matrix

for(i=0;i<n;i++)< span="" style="box-sizing: border-box;">

{

for(j=0;j<r;j++)< span="" style="box-sizing: border-box;">

{

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

}

}

printf("\n");

while(flag)

{

flag=0;

for(i=0;i<n;i++)< span="" style="box-sizing: border-box;">

{

int c=0;

for(j=0;j<r;j++)< span="" style="box-sizing: border-box;">

{

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

{

c++;

if(c==r)

{

for(k=0;k<r;k++)< span="" style="box-sizing: border-box;">

{

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

finish[i]=1;

flag=1;

}

printf("P%d->",i);

if(finish[i]==1)

{

i=n;

}

}

}

}

}

}

for(i=0;i<n;i++)< span="" style="box-sizing: border-box;">

{

if(finish[i]==1)

{

c1++;

}

else

{

printf("P%d->",i);

}

}

if(c1==n)

{

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

}

else

{

printf("\n Process are in dead lock");

printf("\n System is in unsafe state");

}

}</n;i++)<></r;k++)<></r;j++)<></n;i++)<></r;j++)<></n;i++)<></n;i++)<></r;j++)<></r;j++)<></r;j++)<></n;i++)<></r;j++)<></r;j++)<></n;i++)<></r;j++)<></n;i++)<>

OUTPUT:

Enter the no of processes 5

Enter the no of resources instances 3

Enter the max matrix

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter the allocation matrix

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Enter available resources 3 2 2

P1->p3->p4->p2->p0->

The system is in safe state.