**Task 8: [CO3]**

**Banker’s algorithm for Dead Lock Avoidance**

**AIM:**

To Simulate bankers algorithm for Dead Lock Avoidance (Banker‘s Algorithm)

**DESCRIPTION:**

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 the 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 the program.

2. Get the values of resources and processes.

3. Get the avail value.

4. After allocation find the need value.

5. Check whether it is 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 the program.

**Program (Banker’s Algorithm)**

#include<stdio.h>

#include<string.h>

void main()

{

int alloc[10][10],max[10][10];

int avail[10],work[10],total[10];

int i,j,k,n,need[10][10];

int m;

int count=0,c=0;

char finish[10];

printf("Enter the no. of processes and resources:");

scanf("%d%d",&n,&m);

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

finish[i]='n';

printf("Enter the claim matrix:\n");

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

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

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

printf("Enter the allocation matrix:\n");

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

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

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

printf("Resource vector:");

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

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

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

avail[i]=0; for(i=0;i<n;i++)

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

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

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

work[i]=avail[i];

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

work[j]=total[j]-work[j];

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

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

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

A:

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

{

c=0;

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

if((need[i][j]<=work[j])&&(finish[i]=='n'))

c++;

if(c==m)

{

printf("All the resources can be allocated to Process %d", i+1);

printf("\n\nAvailable resources are:");

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

{

work[k]+=alloc[i][k];

printf("%4d",work[k]);

}

printf("\n");

finish[i]='y';

printf("\nProcess %d executed?:%c \n",i+1,finish[i]);

count++;

}

}

if(count!=n)

goto A;

else

printf("\n System is in safe mode");

printf("\n The given state is safe state");

}

**OUTPUT (Banker’s Algorithm)**

Enter the no. of processes and resources: 4 3

Enter the claim matrix:

3 2 2

6 1 3

3 1 4

4 2 2

Enter the allocation matrix:

1 0 0

6 1 2

2 1 1

0 0 2

Resource vector:9 3 6

All the resources can be allocated to Process 2

Available resources are: 6 2 3

Process 2 executed?:y

All the resources can be allocated to Process 3 Available resources

are: 8 3 4

Process 3 executed?:y

All the resources can be allocated to Process 4 Available resources

are: 8 3 6

Process 4 executed?:y

All the resources can be allocated to Process 1

Available resources are: 9 3 6

Process 1 executed?:y

System is in safe mode

The given state is safe state