**Student Name:** Raja Singh Yadav

**Student ID:** 11616181

**Email-id:** Sraja4063@gmail.com

**Github:** **https://github.com/Sraj4063/os\_project\_CA2**

**Description:**

Bankers Algorithm is a “Deadlock Avoidance” algorithm. It is further divided into resource allocation and safety algorithm.

Before proceeding to the algorithm, coming to the concept of “Avoidance” which means that we bring into consideration “beforehand” the resources available, analyzing the maximum resources needed by a process and calculating its need from the already allocated resources. Then the requested resources are allocated only in case that after its allocation all the other resources are able to complete their execution.

To solve this problem let us assume that there are n processes and m resources. So following are the parameters in it:-

1.Maximum (Max): -There is n\*m matrix in which there are maximum number of instances of a resource a process can acquire i.e. if Max[i][j] =k, it means that a process P(i) can have only k instances of resource type R(j).

2.Available:-It represents the available resources of each type i.e. if Available[j]=k, it means that there are k instances of resource type R(j).

3.Allocation:-It represents the number of resources of each type is currently allocated i.e. Allocation[i][j] = k, then there are k instances of resource R(j) allocated to process P(i).

4.Need:-It represents resources needed by process i.e. Need[i][j] = k, then k instances are needed.

Algorithm:

Resource allocation algorithm:

1. If Request i<=Need i, go to step 2. Else the request is not valid.

2. If Request i<=Available i, go to step 3, else the Pi must wait, since resources are not available.

3. Pretend to allocate requested resources to Pi by modifying the state as follows:

Available: = Available +Request I;

Allocation i: = Allocation I +Request I;

Need i: =Need i-Request i;

\*If safe the resources are allocated to Pi.

\*If unsafe Pi must wait, and the old resource-allocation state is restored.

Safety Algorithm:

1.Let Work and Finish Vectors of length m and n respectively. Initialise:

Work: =Available

Finish[i]=false for i=0, 1, 2 , …..,n

2.) Find i such that both:

a) Finish[i]=false

b) Need i<=work

If no such i exists, go to step 4.

3.) Work: =Work + Allocated i

Finish[i]:=true

Go to step 2.

4.) If Finish[i]=true for all I,then the system is in a safe state.

**Code:**

#include <stdio.h>

int allocatted[5][5], max\_needs[5][5], available[5];

int allocation[5] = {0, 0, 0, 0, 0};

int maximum\_res[5], running[5], safe = 0;

int count = 0, i, j, exec, resource, process, k = 1;

int main()

{

printf("\nEnter number of process: ");

scanf("%d", &process);

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

{

running[i] = 1;

count++;

}

printf("\nEnter number of resource: ");

scanf("%d", &resource);

printf("\nEnter Claim Vector:");

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

{

scanf("%d", &maximum\_res[i]);

}

printf("\nEnter allocatted Resource Table:\n");

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

{

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

{

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

}

}

printf("\nEnter Maximum Claim Table:\n");

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

{

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

{

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

}

}

printf("\nThe Claim Vector is: ");

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

{

printf("\t%d", maximum\_res[i]);

}

printf("\nThe allocatted Resource Table:\n");

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

{

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

{

printf("\t%d", allocatted[i][j]);

}

printf("\n");

}

printf("\nThe Maximum Claim Table:\n");

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

{

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

{

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

}

printf("\n");

}

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

{

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

{

allocation[j] += allocatted[i][j];

}

}

printf("\nallocatted resource:");

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

{

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

}

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

{

available[i] = maximum\_res[i] - allocation[i];

}

printf("\nAvailable resource:");

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

{

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

}

printf("\n");

while (count != 0)

{

safe = 0;

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

{

if (running[i])

{

exec = 1;

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

{

if (max\_needs[i][j] - allocatted[i][j] > available[j])

{

exec = 0;

break;

}

}

if (exec)

{

printf("\nProcess%d is executing\n", i + 1);

running[i] = 0;

count--;

safe = 1;

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

{

available[j] += allocatted[i][j];

}

break;

}

}

}

if (!safe)

{

printf("\nThe process are in unsafe state.\n");

break;

}

else

{

printf("\nThe process is in safe state");

printf("\nAvailable vector:");

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

{

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

}

printf("\n");

}

}

return 0;

}

**Complexity:**

O( n2) because there is a ‘for’ loop in a ‘while’ loop.

**Boundary conditions:**

1.If request>need or request>available , the resources will not be granted to the process.

2.For the above code the resources entered can be from 0 to 9.

**Test cases:**

Allocated Maximum Available

A B C A B C A B C

P0 0 0 1 8 4 3 3 2 2

P1 3 2 0 6 2 0

P2 2 1 1 3 3 3

Req:

P0 0 0 2

A B C Req<Available

P0 8 4 2 Available=Available-Req=322-002=320

P1 3 0 0 Allocated=Allocated+Req=001+002=003

P2 1 2 2 Need=Need-Req=842-002=840

New Need

A B C

P0 8 4 0 8 4 0>3 2 0 false

P1 3 0 0 3 0 0<3 2 0 true Available=320+320=640

P2 1 2 2 1 2 2<6 4 0 true Available=640+211=851

P0 8 4 0<8 5 1 true Available=851+003=854

Therefore , all the processes are able to execute if the request is granted.

OUTPUT=>Process is in safe state.

Req2:A B C

P1 2 0 0

Similarly,Req<Need and Available , so we get , Available=322-200=122

Alloacted=320+200=520

Need=300-200=100

Need

A B C

P0 8 4 2 8 4 2>1 2 2 false

P1 1 0 0 1 0 0<1 2 2 true Available=122+520=640

P2 1 2 2 1 2 2<6 4 0 true Availble=640+211=853

P0 8 4 2<8 5 3 true Available=853+001=854

Therefore,all processes are able to execute.

OUTPUT=>Process is in safe state.