Name – Disha Arora

Sec – KI8GE

RegNo. – 11806276

Sub – Operating System

**BANKERS ALGORITHM**

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

Following **Data structures** are used to implement the Banker’s Algorithm:

Let **‘n’**be the number of processes in the system and **‘m’**be the number of resources types.

**Available:**

* It is a 1-d array of size **‘m’** indicating the number of available resources of each type.
* Available[ j ] = k means there are **‘k’** instances of resource type **Rj**

**Max:**

* It is a 2-d array of size ‘**n\*m’**that defines the maximum demand of each process in a system.
* Max [ i, j ] = k means process **Pi** may request at most **‘k’** instances of resource type **Rj.**

**Allocation:**

* It is a 2-d array of size**‘n\*m’**that defines the number of resources of each type currently allocated to each process.
* Allocation[ i, j ] = k means process **Pi** is currently allocated **‘k’** instances of resource type **Rj**

**Need:**

* It is a 2-d array of size **‘n\*m’** that indicates the remaining resource need of each process.
* Need [ i,   j ] = k means process **Pi** currently need **‘k’** instances of resource type **Rj**
* Need [ i,   j ] = Max [ i,   j ] – Allocation [ i,   j ]

Allocationi specifies the resources currently allocated to process Pi and Needi specifies the additional resources that process Pi may still request to complete its task.

Banker’s algorithm consists of Safety algorithm and Resource request algorithm:

**Safety Algorithm**

The algorithm for finding out whether or not a system is in a

safe state can be described as follows:

*Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.*

*1) Initialize: Work = Available  
Finish[i] = false; for i=1, 2, 3, 4….n*

*2) Find an i such that both  
a) Finish[i] = false  
b) Needi <= Work  
if no such i exists goto step (4)*

*3) Work = Work + Allocation[i]  
Finish[i] = true  
goto step (2)*

*4) if Finish [i] = true for all i  
then the system is in a safe state*

**Resource-Request Algorithm**

Let Requesti be the request array for process Pi. Requesti[j] = k means process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:

*1) If Requesti <= Needi  
Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.*

*2) If Requesti <= Available  
Goto step (3); otherwise, Pi must wait, since the resources are not available.*

*3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as  
follows:  
Available = Available – Requesti  
Allocationi = Allocationi + Requesti  
Needi = Needi– Requesti*

**PROGRAM FOR EXECUTION**

#include<conio.h>

#include<stdio.h>

int main()

{

int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10],newres[10];

int p, r, i, j, process, count;

count = 0;

printf("Enter the no of processes : ");

scanf("%d", &p);

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

completed[i] = 0;

printf("\n\nEnter the no of resources : ");

scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");

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

{

printf("\nFor process %d : ", i + 1);

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

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

}

printf("\n\nEnter the allocation for each process : ");

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

{

printf("\nFor process %d : ",i + 1);

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

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

}

printf("\n\nEnter the Available Resources : ");

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

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

printf("\n\nEnter the newres for each process : ");

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

{

printf("\nFor process %d : ", i + 1);

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

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

}

avail[i]=avail[i] - newres[i];

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

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

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

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

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

do

{

printf("\n Max matrix:\tAllocation matrix:\n");

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

{

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("\n");

}

process = -1;

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

{

if(completed[i] == 0)//if not completed

{

process = i ;

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

{

if(avail[j] < need[i][j])

{

process = -1;

break;

}

}

}

if(process != -1)

break;

}

if(process != -1)

{

printf("\nProcess %d runs to completion!", process + 1);

safeSequence[count] = process + 1;

count++;

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

{

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

alloc[process][j] = 0;

Max[process][j] = 0;

completed[process] = 1;

}

}

}while(count != p && process != -1);

if(count == p)

{

printf("\nThe system is in a safe state!!\n");

printf("Safe Sequence : < ");

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

printf("%d ", safeSequence[i]);

printf(">\n");

}

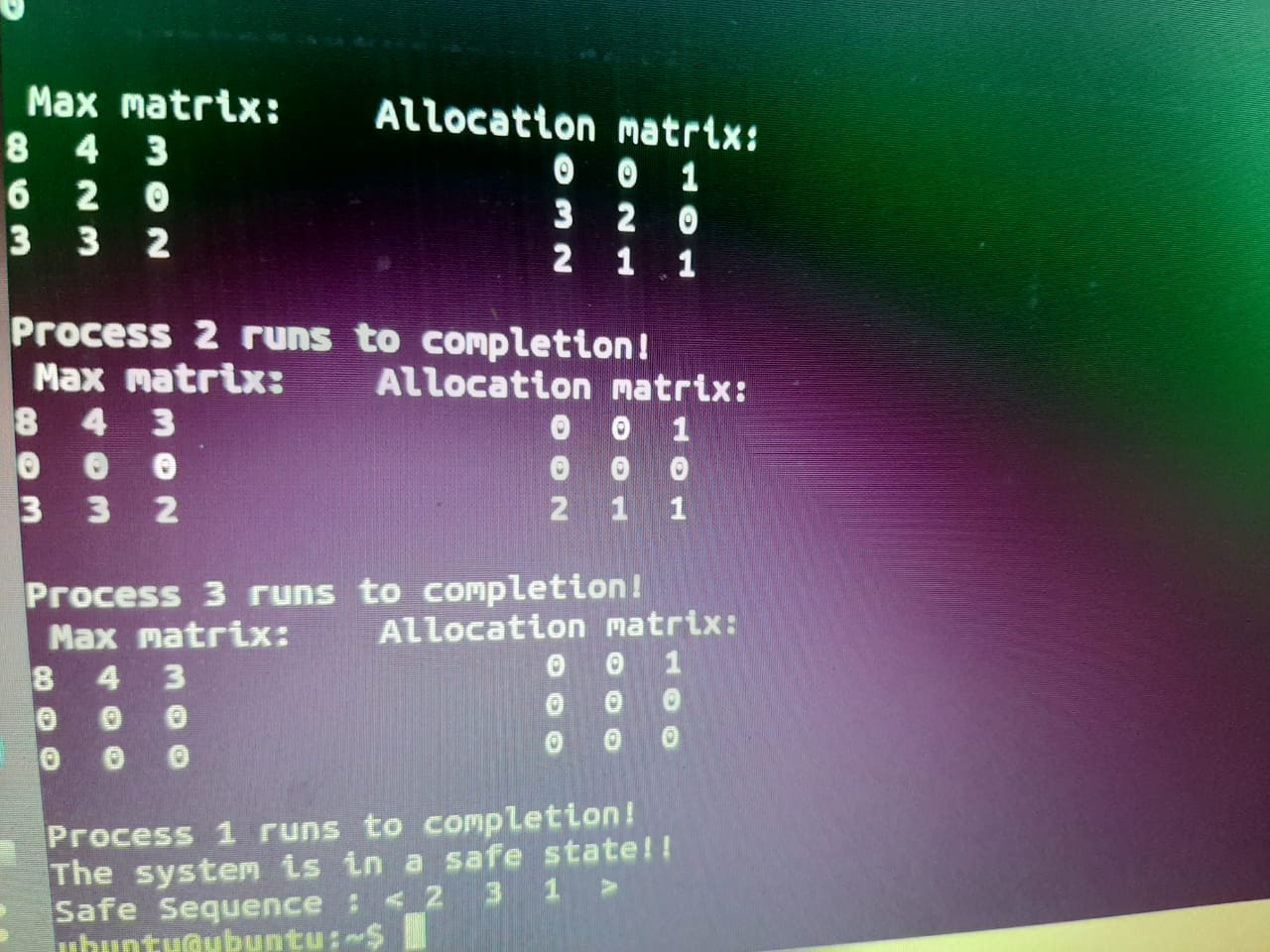
else

printf("\nThe system is in an unsafe state!!");

getch();

}

**EXECUTION**



**EXPLANATION**

* We have different variables of integer type.

Max –it will tell the max instances of different resources used by each process.

Alloc – it will tell the instances of different resources allocated to each process.

Avail – it will tell the instance of resource available to fulfill the need of each process.

Newres – it will tell the request resource of each process.

Need – it will tell the need of instance of each process. It is calculated by the formula: Need = Max – Alloc.

p - it defines the number of process.

r - it defines the number of resource.

Process - it is a variable initialized as -1.

Completed- it is initialized as 0. It will define that process execution is complete or not.

If its value will change to one it will indicate that process is completed.

Count – it a counter variable which is initialized to 0 and will increase its value when a process will be completed.

After the declaration of variables:

* We will ask the user for input of each variable.

For the input of resource request if we donot have resource request for any process, we can give 0 as iput of that process.

After taking input of resource request available and allocated matrix will be changed by the formula:

Avail = avail – newres

Alloc = alloc + newres

Using two for loops

Need will be calculated for each process .Allocated and max matrix will also be displayed on the output screen.

* Do while loop will be used.

Process will be initialed as -1.

We will use for loop to check each process. Under that loop we will use if condition to check that a process is completed or not by the condition completed[i]=0.

If this condition is true. For loop for resources will be used and under that if condition will be used to check that avail is greater than need or not avail[j] <need[i][j]. If this condition is true process will again become -1.

* If process!= -1 then safe sequence of that process number will be generated. Value of count will be incremented.

For that process:

Avail=avail+alloc

Max =0

Need=0

Its completed status will be changed to 1

Completed[i]=1.

All the process will checked one by one and computer will do same task till while condition is true that count ! = total no of processes and process status is -1.

Count!=p && process= -1.

* If count==p then system will be in safe state and safe sequence will be printed on the output screen.

If this condition is false system will be in unsafe state and same will be printed on the output screen.