**Experiment No. 3**

**Aim**: Implementation of Banker’s Algorithm

**Theory:** The **Banker's algorithm** is a resource allocation and deadlock avoidance algorithm developed by [Edsger Dijkstra](http://en.wikipedia.org/wiki/Edsger_Dijkstra) that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes a "safe-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.

**Algorithm: Data Structures for the Banker’s Algorithm**

1. Let *n* = number of processes, and *m* = number of resources types.
2. **Available***:* Vector of length *m*.

If available [*j*] = *k*, there are *k* instances of resource type *Rj*available

1. **Max***: n x m* matrix.

If *Max* [*i,j*] = *k*, then process *Pi* may request at most *k* instances of resource type *Rj*

1. **Allocation***: n* x *m* matrix.

If Allocation[*i,j*] = *k* then *Pi* is currently allocated *k* instances of *Rj*

1. **Need***: n* x *m* matrix.

If *Need*[*i,j*] = *k*, then *Pi* may need *k* more instances of *Rj*to complete its task  
 *Need* [*i,j]* = *Max*[*i,j*] – *Allocation* [*i,j*]

**Safety algorithm**

Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize:

1. *Work* = *Available*

*Finish* [*i*] = *false* for *i* = 0, 1, …, *n-* 1

2. Find an *i* such that both:

(a) *Finish* [*i*] = *false*

(b) *Needi* ? *Work*

If no such *i* exists, go to step 4

*3. Work* = *Work* + *Allocationi*  
 *Finish*[*i*] = *true*  
 go to step 2

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

Code:

Output:

Conclusion

Viva questions:

Solve one numerical of banker’s algorithm

**CODE:**

#include<stdio.h>

int main()

{

int k = 0,op, output[10], d = 0, t = 0, ins[5], i, avail[5], allocated[10][5], need[10][5], MAX[10][5], pno, P[10], j, rz,rz1, count = 0;

printf("\nEnter the number of processes:");

scanf("%d", &pno);

op=pno;

printf("\nEnter the number of resources: ");

scanf("%d", &rz);

rz1=rz;

printf("Enter the max instances of each resources: \n");

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

{

avail[i] = 0;

printf("%c= ", (i + 65));

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

}

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

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

printf("\t%c", (i + 65));

printf("\n");

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

{

P[i] = i;

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

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

{

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

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

}

}

printf("\nEnter the Max matrix: \n ");

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

{

printf("\t%c", (i + 65));

avail[i] = ins[i] - avail[i];

}

printf("\n");

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

{

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

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

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

}

printf("\n");

printf("\nEnter the available resources: \n ");

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

{

printf("%c ", (i + 65));

}

printf("\n");

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

{

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

}

while(d!=-1){

d=-1;

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

{

count = 0;

t = P[i];

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

{

need[t][j] = MAX[t][j] - allocated[t][j];

if (need[t][j] <= avail[j])

count++;

}

if (count == rz)

{

output[k++] = P[i];

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

avail[j] += allocated[t][j];

}

else

P[++d] = P[i];

}

pno = d + 1;

}

printf("Need Matrix: \n");

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

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

for(j=0;j<rz1;j++)

{ need[i][j] = MAX[i][j] - allocated[i][j];

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

}

printf("\n");

}

printf("The safe sequence is : \n");

printf("\t <");

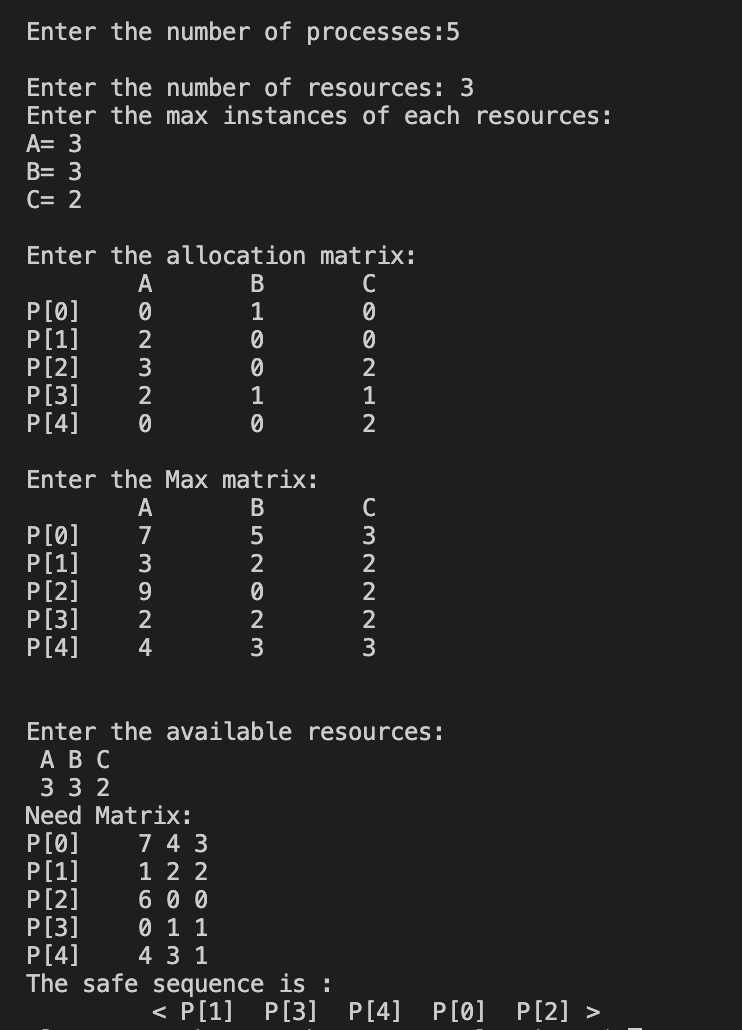
for (i = 0;i < k;i++)

printf(" P[%d] ", output[i]);

printf(">\n");

}

**OUTPUT:**

****