AIM: To implement the bankers algorithm for Deadlock avoidance

Description:

The Bankers Algorithm consists of 2 Algorithms:

1: Safety algorithm :

1) Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.
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) Need$_i$ <= 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

2: Resource Request Algorithm:

1) If Request$_i$ <= Need$_i$
Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If Request$_i$ <= Available
Goto step (3); otherwise, P$_i$ 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
Allocation$_i$ = Allocation$_i$ + Request$_i$
Need$_i$ = Need$_i$– Request$_i$

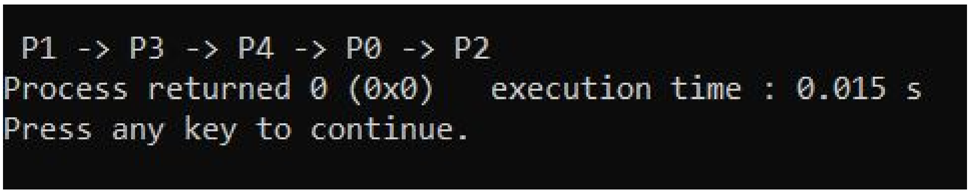
Source Code:

```
#include <stdio.h>

int main()
```

```
{
        int n, m, i, j, k;
        n = 5;
        m = 3;
        int alloc[5][3] = { { 0, 1, 0 },{ 2, 0, 0 },{ 3, 0, 2 },
{ 2, 1, 1 },{ 0, 0, 2 } };
        int max[5][3] = { { 7, 5, 3 },
                                { 3, 2, 2 },
                                { 9, 0, 2 },
                                { 2, 2, 2 }, { 4, 3, 3 } };

        int avail[3] = { 3, 3, 2 };
        int f[n], ans[n], ind = 0;
        for (k = 0; k < n; k++) {
                f[k] = 0;
        }
        int need[n][m];
        for (i = 0; i < n; i++) {
                for (j = 0; j < m; j++)
                        need[i][j] = max[i][j] - alloc[i][j];
        }
        int y = 0;
        for (k = 0; k < 5; k++) {
                for (i = 0; i < n; i++) {
                        if (f[i] == 0) {

                                int flag = 0;
                                for (j = 0; j < m; j++) {
                                        if (need[i][j] > avail[j]){
                                                flag = 1;
                                                break;
```

```
                    }
                }

                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }

    printf("\n");
     for (i = 0; i < n - 1; i++)
  printf(" P%d ->", ans[i]);
 printf(" P%d", ans[n - 1]);
    return (0);
}
```

Output:



```
 P1 -> P3 -> P4 -> P0 -> P2
Process returned 0 (0x0)   execution time : 0.015 s
Press any key to continue.
```

Result Analysis:

We hard code the maximum allocation, available and the currently allocated resources for 5 processes and the number of resources are 3. As we can see, resources can be allocated and a safe sequence exists. So we can go ahead with having these 5 processes execute without deadlocks

References:

Www.google.com

Www.geeksgforgeeks.com

www.thecrazyprogrammer.com