Assignment - 03

Title: Bankers algorithm for deadlock avoidance

Course Title: Operating system

Course Code: CSE31P4

Submitted To

Moshiur Rahman
Associate Professor
Computer Science and Engineering
Bangladesh Open University

Submitted By

Tamima Yeasmin
ID: 17-0-52-801-003
Third Year First Semester
Session: 2017-2018

Date of Assignment: 09/07/21

Date of Submission: 06/08/21

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.

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length 'm' and 'n' resp ectively.
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

For example: Safe Sequence (Determine) Processes - P0 P1 P2 P3 P4

Process	Allocation A B C (Memory Block)			Max A B C			Available (Work) A B C			Need A B C = Max - Allocation		
P0	0	1	0	7	5	3	~~3~~	~~3~~	~~2~~	7	4	3
P1	2	0	0	3	2	2	~~5~~	~~3~~	~~2~~	1	2	2
P2	3	0	2	9	0	2	~~7~~	~~4~~	~~3~~	6	0	0
P3	2	1	1	2	2	2	~~7~~	~~4~~	~~5~~	0	1	1
P4	0	0	2	4	3	3	~~7~~	~~5~~	~~5~~	4	3	1
							10	5	7			

Answer:

According to banker’s algorithm,

If [Need ≤ Work (Available)] is true, then we have to update Work = Work + Allocation

For P0 => 7 4 3 ≤ 3 3 2 False

For P1 => 1 2 2 ≤ 3 3 2 True W = W + A

= 332 + 200

= 532

For P2 => 6 0 0 ≤ 5 3 2 False

For P3 => 0 1 1 ≤ 5 3 2 True W = W + A

= 532 + 211

= 743

For P4 => 4 3 1 ≤ 7 4 3 True   W = W + A

= 743 + 002

= 745

For P0 => 7 4 3 ≤ 7 4 5 True   W = W + A

= 745 + 010

= 755

For P2 => 6 0 0 ≤ 7 5 5 True   W = W + A

= 755 + 302

= 1057

So, the safe sequence P1, P3, P4, P0, P2

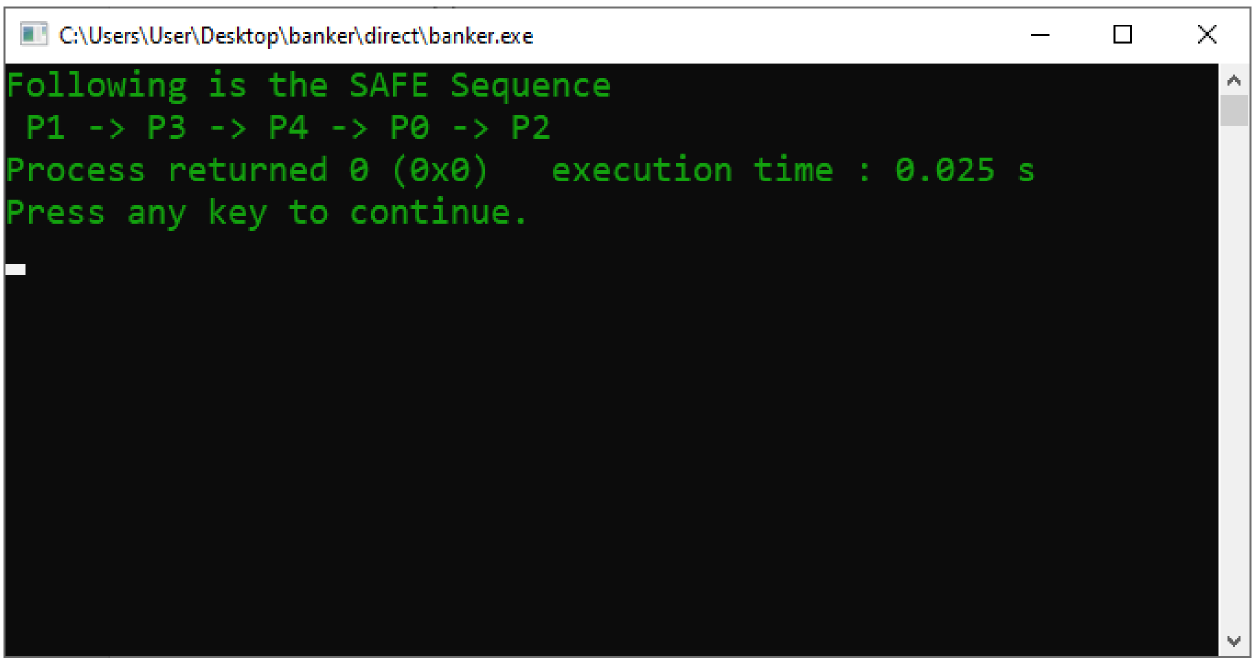
Source Code for direct answer:

```
// Banker's Algorithm
#include <stdio.h>
int main()
{
	// P0, P1, P2, P3, P4 are the Process names here

	int n, m, i, j, k;
	n = 5; // Number of processes
	m = 3; // Number of resources
	int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
						{ 2, 0, 0 }, // P1
						{ 3, 0, 2 }, // P2
						{ 2, 1, 1 }, // P3
						{ 0, 0, 2 } }; // P4
```

```
int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
                  { 3, 2, 2 }, // P1
                  { 9, 0, 2 }, // P2
                  { 2, 2, 2 }, // P3
                  { 4, 3, 3 } }; // P4

int avail[3] = { 3, 3, 2 }; // Available Resources

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("Following is the SAFE Sequence\n");
	for (i = 0; i < n - 1; i++)
		printf(" P%d ->", ans[i]);
	printf(" P%d", ans[n - 1]);

	return (0);

	// This code is contributed by Deep Baldha (CandyZack)
}
```

Output:



Source Code for indirect answer:

```
// Banker's Algorithm
#include <stdio.h>
int main()
{
    // P0, P1, P2, P3, P4 are the Process names here

    int n, m, i, j, k;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    //n = 5; // Number of processes

    printf("Enter the number of resources: ");
    scanf("%d", &m);
    //m = 3; // Number of resources

    printf("Enter value of allocation: \n");
    int alloc[n][m];
    for(i=0; i<n; i++)
        for(j=0; j<m; j++)
            scanf("%d", &alloc[i][j]);

    /*int alloc[5][3] = { { 0, 1, 0 }, // P0    // Allocation Matrix
                        { 2, 0, 0 }, // P1
                        { 3, 0, 2 }, // P2
                        { 2, 1, 1 }, // P3
                        { 0, 0, 2 } }; // P4*/

    printf("Enter value of maximum: \n");
    int max[n][m];
    for(i=0; i<n; i++)
        for(j=0; j<m; j++)
            scanf("%d", &max[i][j]);


    /*int max[5][3] = { { 7, 5, 3 }, // P0    // MAX Matrix
                      { 3, 2, 2 }, // P1
                      { 9, 0, 2 }, // P2
                      { 2, 2, 2 }, // P3
                      { 4, 3, 3 } }; // P4*/
```

```
int avail[3]; //= { 3, 3, 2 }; // Available Resources
printf("Enter value of available: \n");
for(i=0; i<m; i++)
    scanf("%d", &avail[i]);

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("Following is the SAFE Sequence\n");
for (i = 0; i < n - 1; i++)
```

```
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);
}
```

Output:

Select C:\Users\User\Desktop\banker\banker.exe

```
Enter the number of processes: 5
Enter the number of resources: 3
Enter value of allocation:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter value of maximum:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter value of available:
3 3 2
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
Process returned 0 (0x0)   execution time : 47.817 s
Press any key to continue.
```

Finally, our given problem’s safe sequence is P1 → P3 → P4 → P0 → P2.

THE END