Banker's Algorithm for Deadlock Avoidance

Nirbhay Sharma, Garima Saigal, Eklavya Sanmotra, and Kumud Kesar

2021a1r092@mietjammu.in

2021a1r094@mietjammu.in

2021a1r101@mietjammu.in

2021a1r112@mietjammu.in

Model Institute of Engineering and Technology, CSE department, Kot Bhalwal, Jammu, Jammu and Kashmir 181122

ACKNOWLEDGMENT

We would like to extend our deep sense of gratitude towards our respected and learned-guide, Assistant professor Ms. Pragti Jamwal for their valuable help and guidance, we are thankful to them for the immense support and encouragement they have given us throughout this project.

We feel immensely proud to extend our heartiest thanks to Dr.Ankur Gupta, director of MIET, for providing us with a platform to improve in various fields and permitting us to utilize all the necessary facilities of the institution.

We are also thankful to all the other faculty and staff members of our department for their kind co-operation and help. Their open-hearted help and counsel have been an absolute source of inspiration.

Report on “Banker's Algorithm for Deadlock Avoidance”

Abstract: Deadlock-free operation is essential for operating highly automated manufacturing systems. The seminal deadlock avoidance procedure, Banker's Algorithm, was developed for computer operating systems, an environment where very little information regarding the future resource requirements of executing processes is known. Manufacturing researchers have tended to dismiss Bankers as being too conservative in the manufacturing environment where future resource requirements are well defined by part routes. In this work, we investigate this issue by developing variants of Banker’s Algorithm that are applicable to buffer space allocation in flexible manufacturing. We show that these algorithms are not overly conservative and that indeed, Banker’s approach can provide very good operational flexibility when properly applied to the manufacturing environment.

INTRODUCTION

Banker's algorithm is a deadlock avoidance algorithm. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not. Consider there are n account holders in a bank and the sum of the money in all of their accounts is S. Every time a loan has to be granted by the bank, it subtracts the loan amount from the total money the bank has. Then it checks if that difference is greater than S. It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once. Banker's algorithm works in a similar way in computers. Whenever a new process is created, it must specify the maximum instances of each resource type that it needs, exactly.

PROBLEM STATEMENT

Implementation of most representative Banker’s Algorithm on BASH Shell for deadlock avoidance.

OBJECTIVE

It is an algorithm used to avoid deadlock and allocate resources safely to each process in the computer system. It examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes. It is named as Banker’s Algorithm because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate the allocation of resources.

IMPLEMENTATION

The requirements of this project are:

1. Linux Operating System
2. GCC
3. Nano Editor
4. Bankers Algorithm

ALGORITHM

Safety Algorithm:

1) Let Work and Finish be Arrays 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) Needi <= Work

if no such i exists goto step (4)

2) 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

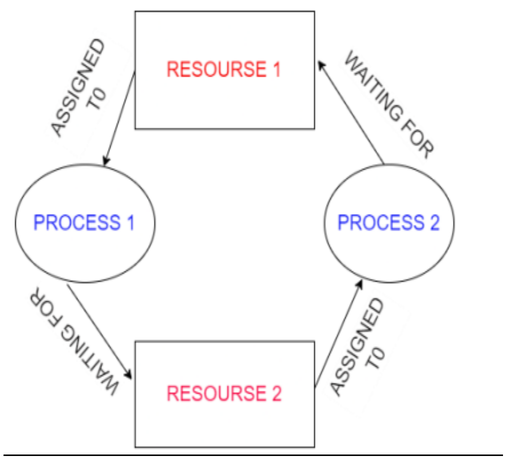
Request Algorithm:

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

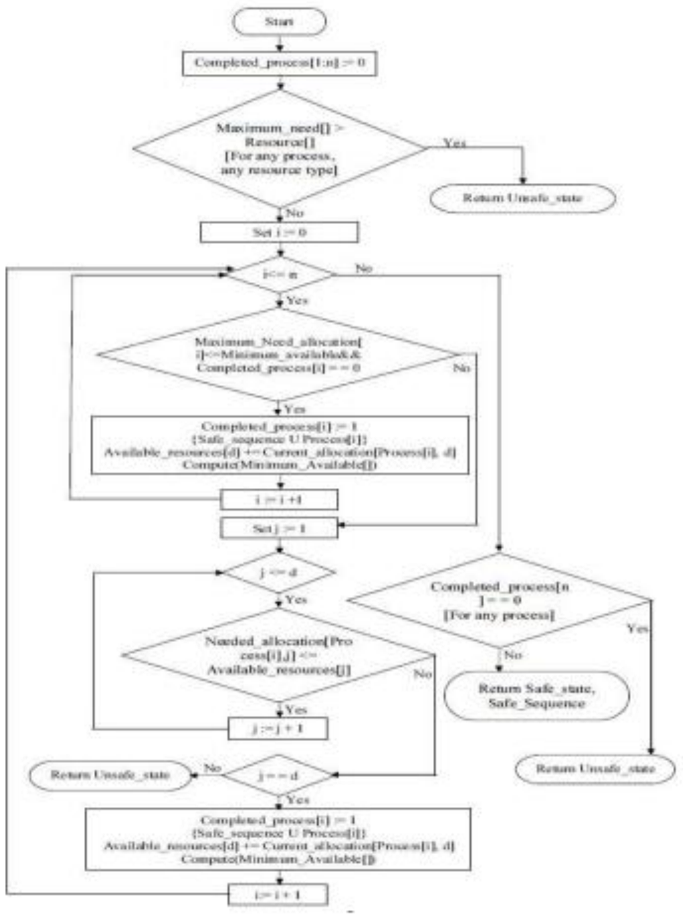
2) If Request i <= Available
   Goto step (3); otherwise, Pi must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:
   Available = Available – Request i
   Allocation i = Allocation i +Request i Need i = Need i– Request i

I/O DIAGRAM



FLOWCHART


Start
Completed_process[1:n] := 0
Maximum_need[] > Resource[] [For any process, any resource type]
Yes
Return Unsafe_state
No
Set i := 0
i<= n
No
Yes
Maximum_Need_allocation[i]<=Minimum_available& Completed_process[i] = = 0
No
Yes
Completed_process[i] := 1 {Safe_sequence U Process[i]} Available_resources[d] += Current_allocation[Process[i], d] Compute(Minimum_Available[])
i := i +1
Set j := 1
j <= d
Yes
Completed_process[n] = = 0 [For any process]
Yes
No
Return Safe_state, Safe_Sequence
Needed_allocation[Process[i],j] <= Available_resources[j]
No
Yes
j := j + 1
Return Unsafe_state
No
j = = d
Yes
Return Unsafe_state
Completed_process[i] := 1 {Safe_sequence U Process[i]} Available_resources[d] += Current_allocation[Process[i], d] Compute(Minimum_Available[])
i:= i + 1

CODE

Implementation of the most representative Banker’s Algorithm on BASH Shell for deadlock avoidance

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

	int n, r, i, j, k;
	n = 5; // Indicates the Number of processes
	r = 3; //Indicates the Number of resources
	int alloc[5][3] = { { 0, 0, 1 }, // P0 // This is Allocation Matrix

	{ 3, 0, 0 }, // P1

	{ 1, 0, 1 }, // P2

	{ 2, 3, 2 }, // P3

	{ 0, 0, 3 } }; // P4

	int max[5][3] = { { 7, 6, 3 }, // P0 // MAX Matrix
						{ 3, 2, 2 }, // P1
						{ 8, 0, 2 }, // P2
						{ 2, 1, 2 }, // P3
						{ 5, 2, 3 } }; // P4

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

	int f[n], ans[n], ind = 0;
	for (k = 0; k < n; k++) {
		f[k] = 0;
	}
	int need[n][r];
	for (i = 0; i < n; i++) {
		for (j = 0; j < r; 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 < r; j++) {
					if (need[i][j] > avail[j]){
	flag = 1;
	break;
					}
				}

				if (flag == 0) {
	ans[ind++] = i;
					for (y = 0; y < r; y++)
	avail[y] += alloc[i][y];
```

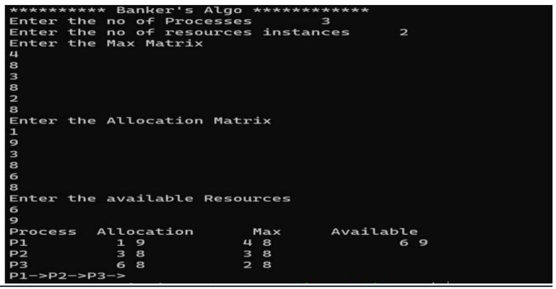
```
                                        f[i] =
1;
                                  }
                            }
                      }
                }


                printf("Th SAFE Sequence is as
follows\n");
                for (i = 0; i < n - 1; i++)
                      printf(" P%d ->", ans[i]);
                printf(" P%d", ans[n - 1]);
else {
 printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
 }
                return (0);


}
```

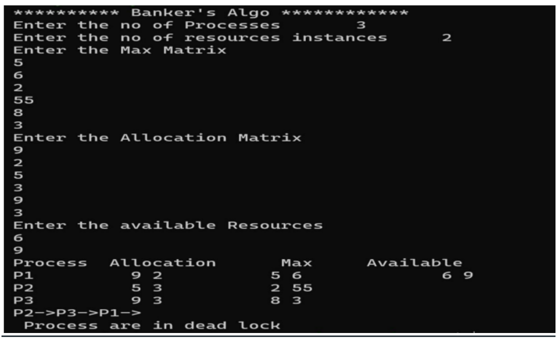
OUTPUT

SAFE STATE



```
********** Banker's Algo ************
Enter the no of Processes          3
Enter the no of resources instances        2
Enter the Max Matrix
4
8
3
8
2
8
Enter the Allocation Matrix
1
9
3
8
6
8
Enter the available Resources
6
9
Process  Allocation      Max      Available
P1          1 9           4 8            6 9
P2          3 8           3 8
P3          6 8           2 8
P1->P2->P3->
```

UNSAFE STATE



```
********** Banker's Algo ************
Enter the no of Processes          3
Enter the no of resources instances        2
Enter the Max Matrix
5
6
2
55
8
3
Enter the Allocation Matrix
9
2
5
3
9
3
Enter the available Resources
6
9
Process  Allocation      Max      Available
P1          9 2           5 6            6 9
P2          5 3           2 55
P3          9 3           8 3
P2->P3->P1->
 Process are in dead lock
```

CONCLUSION AND FUTURE WORK

In conclusion, the Banker's Algorithm is a valuable tool for avoiding deadlocks in a system that allocates resources to multiple processes. By keeping track of the available resources and the maximum resources needed by each process, the algorithm ensures that no process exceeds its resource needs and that no deadlocks occur. The Banker's Algorithm is widely used in operating systems and other software that manage resources in a multi-tasking environment and has proven to be an effective method for ensuring the efficient and deadlock-free allocation of resources. While the algorithm is not foolproof and can still lead to deadlocks in certain situations, it is a valuable tool for avoiding deadlocks in many cases.

REFERENCES

1. http://www.geekforgeeks.com/
2. https://www.studytonight.com/operating-system/bankers-algorithm
3. https://chat.openai.co/
4. http://www.researchgate.com/
5. http://www.javapoint.com/
6. http://www.programiz.com/
7. http://suraj1693.blogspot.com/2013/11/program-for-bankers-algorithm-for.html