Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

**Artificial Intelligence and Data Science Department.**

OS / Even Sem 2021-22 / Experiment 9.

---

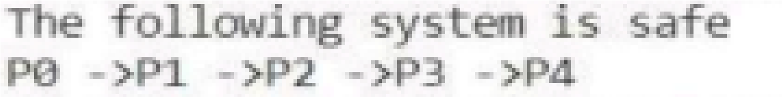
YASH SARANG.
47 / D6AD.
EXPERIMENT - 9.
Banker's Algorithm.

---

**Aim:** WAP to simulate Banker's Algorithm.

---

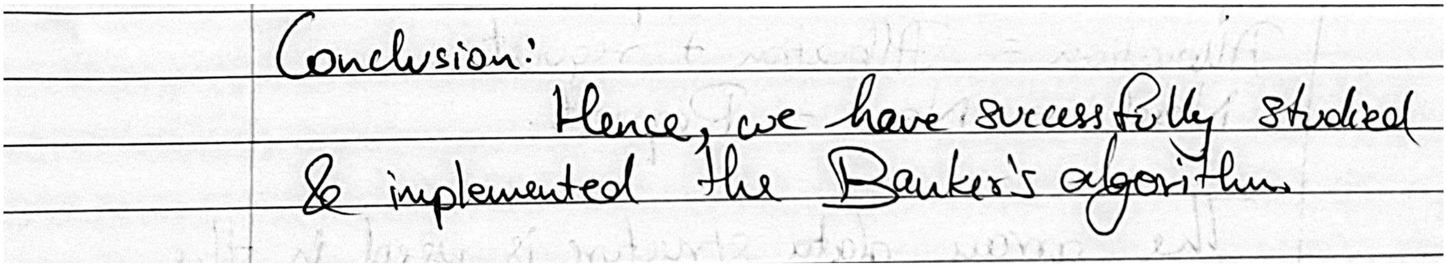
**Output:**



```
The following system is safe
P0 ->P1 ->P2 ->P3 ->P4
```

---

**Conclusion:-**



Conclusion:
Hence, we have successfully studied & implemented the Banker's algorithm.

---

**Theory:**

# EXPERIMENT 9

Aim: Write a program to simulate Banker's Algorithm

Theory:

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

It is used in banking systems to check whether loan is to be sanctioned or not. It consists of a safety algorithm and resources request algorithm.

Safety algorithm is used to find out whether or not a system is in a safe state.

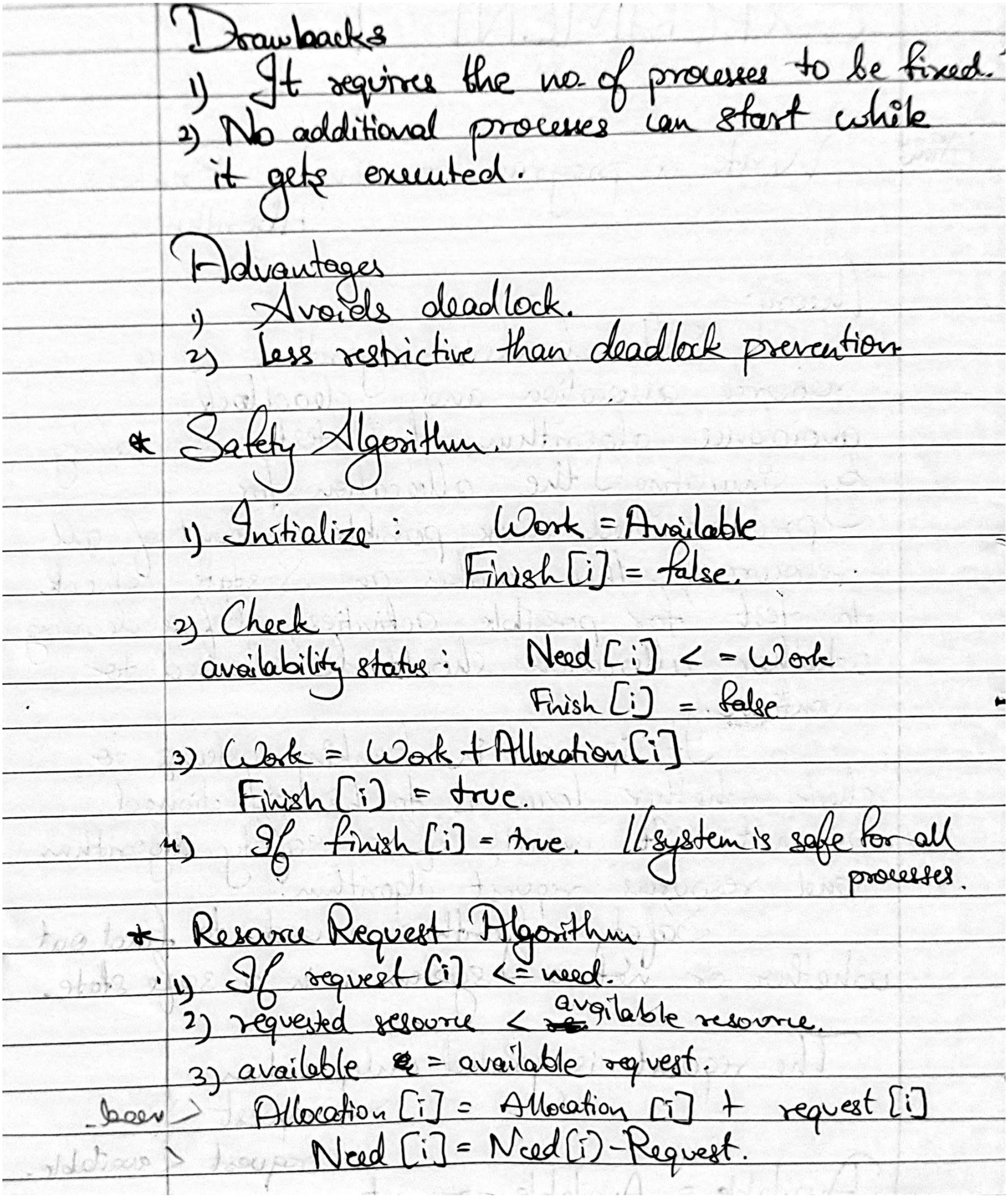
The request is grated only when

request < need

request < available.

Available = Available - request
Allocation = Allocaton + request
Need = Need - Request.

The array data structure is used in the Banker's algorithm.



Drawbacks

1) It requires the no. of processes to be fixed.

2) No additional processes can start while it gets executed.

Advantages

1) Avoids deadlock.

2) Less restrictive than deadlock prevention

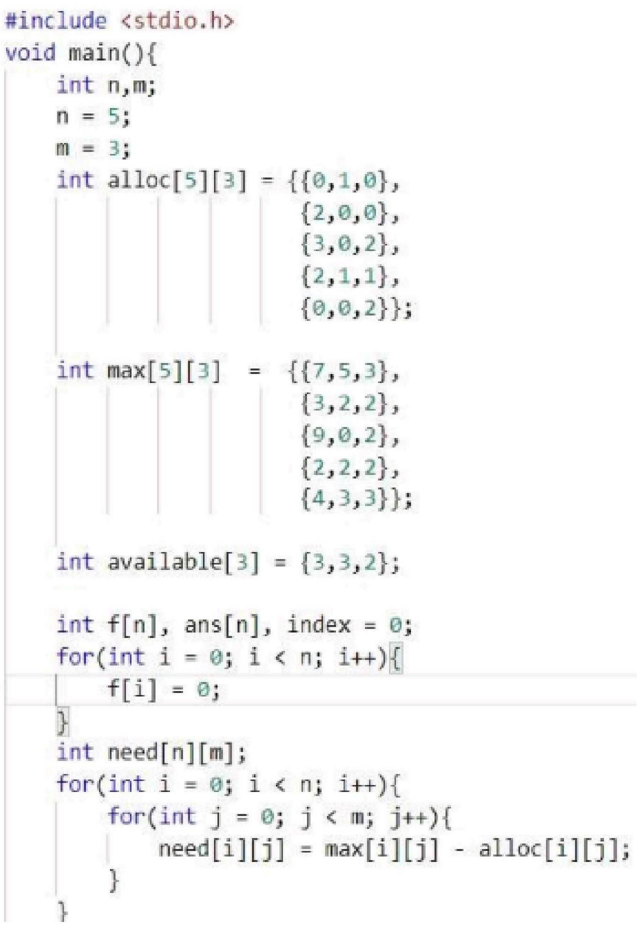
* Safety Algorithm.

```
1) Initialize :        Work = Available
                       Finish [i] = false.
2) Check
   availability status :  Need [i] < = Work
                          Finish [i] = false
3) Work = Work + Allocation [i]
   Finish [i] = true.
4) If finish [i] = true.   // system is safe for all
                                          processes.
```

* Resource Request Algorithm

```
1) If request [i] <= need.
2) requested resource < available resource.
3) available = available request.
   Allocation [i] = Allocation [i] + request [i]
   Need [i] = Need (i) - Request.
```

**Code:**

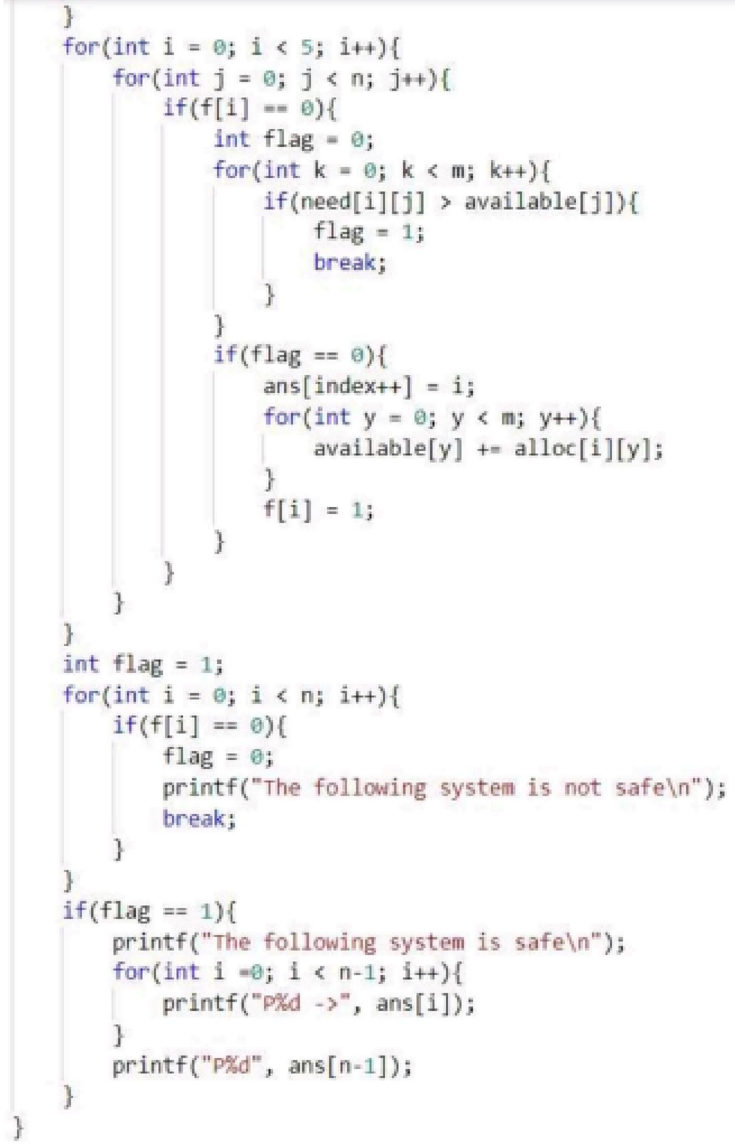


```c
#include <stdio.h>
void main(){
    int n,m;
    n = 5;
    m = 3;
    int alloc[5][3] = {{0,1,0},
                       {2,0,0},
                       {3,0,2},
                       {2,1,1},
                       {0,0,2}};

    int max[5][3]  =  {{7,5,3},
                       {3,2,2},
                       {9,0,2},
                       {2,2,2},
                       {4,3,3}};

    int available[3] = {3,3,2};

    int f[n], ans[n], index = 0;
    for(int i = 0; i < n; i++){
        f[i] = 0;
    }
    int need[n][m];
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }
```



```c
    }
    for(int i = 0; i < 5; i++){
        for(int j = 0; j < n; j++){
            if(f[i] == 0){
                int flag = 0;
                for(int k = 0; k < m; k++){
                    if(need[i][j] > available[j]){
                        flag = 1;
                        break;
                    }
                }
                if(flag == 0){
                    ans[index++] = i;
                    for(int y = 0; y < m; y++){
                        available[y] += alloc[i][y];
                    }
                    f[i] = 1;
                }
            }
        }
    }
    int flag = 1;
    for(int i = 0; i < n; i++){
        if(f[i] == 0){
            flag = 0;
            printf("The following system is not safe\n");
            break;
        }
    }
    if(flag == 1){
        printf("The following system is safe\n");
        for(int i =0; i < n-1; i++){
            printf("P%d ->", ans[i]);
        }
        printf("P%d", ans[n-1]);
    }
}
```