K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: C2	**Roll No.: 16010122323**
Experiment No. 08	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

TITLE: Simulate Bankers Algorithm for Deadlock Avoidance

AIM: Implementation of Banker's Algorithm for Deadlock Avoidance

---

Expected Outcome of Experiment:

CO 3. To understand the concepts of process synchronization and deadlock.

Books/ Journals/ Websites referred:

1. **Silberschatz A., Galvin P., Gagne G. "Operating Systems Principles", Willey Eight edition.**
2. **Achyut S. Godbole , Atul Kahate "Operating Systems" McGraw Hill Third Edition.**
3. **William Stallings, "Operating System Internal & Design Principles", Pearson.**
4. **Andrew S. Tanenbaum, "Modern Operating System", Prentice Hall.**

Pre Lab/ Prior Concepts:

Knowledge of deadlocks and all deadlock avoidance methods.

<u>Description of the application to be implemented</u>:

The Banker's algorithm is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra.

DATA STRUCTURES

(where *n* is the number of processes in the system and *m* is the number of resource types)

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[j] = k means there are **'k'** instances of resource type $\mathbf{R_j}$ **Max:**
- It is a 2-d array of size '**n*m**' that defines the maximum demand of each process in a system.
- Max [i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

Allocation:

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation [i, j] = k means process $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$ **Need:**
- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process $\mathbf{P_i}$ currently need **'k'** instances of resource type $\mathbf{R_j}$
- **Need [i, j] = Max [i, j] – Allocation [i, j]**

Implementation details:

```
n = int(input("Enter number of processes: "))
m = int(input("Enter number of resources: "))

alloc = []
print("Enter allocation matrix: ")
for i in range(n):
    temp = list(map(int, input().split()))
    alloc.append(temp)

max = []
print("\nEnter max matrix: ")
for i in range(n):
    temp = list(map(int, input().split()))
    max.append(temp)

avail = list(map(int, input("\nEnter available resources: 
").split()))

f = [0] * n
ans = [0] * n
ind = 0

need = [[0 for _ in range(m)] for _ in range(n)]
for i in range(n):
    for j in range(m):
        need[i][j] = max[i][j] - alloc[i][j]

for k in range(n):
    for i in range(n):
        if f[i] == 0:
            flag = 0
            for j in range(m):
                if need[i][j] > avail[j]:
                    flag = 1
                    break

            if flag == 0:
```

```
                ans[ind] = i
                ind += 1
                for y in range(m):
                    avail[y] += alloc[i][y]
                f[i] = 1

if ind == n:
    print("\nFollowing is the SAFE Sequence:")
    for i in range(n - 1):
        print(f" P{ans[i]} ->", end="")
    print(f" P{ans[n - 1]}")
else:
    print("The system is NOT in a safe state!")
```

```
Enter number of processes: 5
Enter number of resources: 3
Enter allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter max matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter available resources: 3 3 2
```

Conclusion:

In this experiment we learned and successfully implemented banker's algorithm for resource allocation and deadlock avoidance.

Post Lab Objective Questions

1) The wait-for graph is a deadlock detection algorithm that is applicable when:
a) All resources have a single instance
b) All resources have multiple instances
c) Both a and b
d) None of the above
Ans:
2) Resources are allocated to the process on non-sharable basis is _
a) Hold and Wait
b) Mutual Exclusion
c) No pre-emption
d) Circular Wait
Ans:
3) Which of the following approaches require knowledge of the system state?
a) Deadlock Detection
b) Deadlock Prevention
c) Deadlock Avoidance
d) All of the above
Ans:

4) Consider a system having 'm' resources of the same type. These resources are shared by 3 processes A, B, C which have peak time demands of 3, 4, 6 respectively. The minimum value of 'm' that ensures that deadlock will never occur is

a) 11
b) 12
c) 13
d) 14

A
Ans:

Post Lab Descriptive Questions

1. Consider a system with total of 150 units of memory allocated to three processes as shown:

Process	Max	Hold
P^1	70	45
P^2	60	40
P^3	60	15

Apply Banker's algorithm to determine whether it would be safe to grant each of the following request. If yes, indicate sequence of termination that could be possible.

1) The P^4 process arrives with max need of 60 and initial need of 25 units.
2) The P^4 process arrives with max need of 60 and initial need of 35 units.

Answer 1)

Process	Allocated	Max	Need	Available
P1	45	70	25	25
P2	40	60	20	
P3	15	60	45	
P4	25	60	35	

Available = 150 - (45 + 40 + 15 + 25) = 25

Finish = [0 0 0 0], Work(W) = 25(Available)

P1: Need<=W, => W = W + A

W = 25 + 45 = 70, Finish = [1 0 0 0]

P2: Need<=W, => W = W + A

W = 70 + 40 = 110, Finish = [1 1 0 0]

P3: Need<=W, => W = W + A

W = 110 + 15 = 125, Finish = [1 1 1 0]

P4: Need<=W, => W = W + A

W = 125 + 25 = 150, Finish = [1 1 1 1]

Hence, safe sequence is {P1, P2, P3, P4}. Therefore, system is in safe state.

Answer 2)

Process Allocated Max Need Available

P1 45 70 25 15

P2 40 60 20

P3 15 60 45

P4 35 60 25

Available = 150 - (45 + 40 + 15 + 35) = 15

Finish = [0 0 0 0], Work(W) = 15(Available)

- P1: Need > W => P1 has to wait
- P2: Need > W => P2 has to wait
- P3: Need > W => P3 has to wait
- P4: Need > W => P4 has to wait

Since all processes went to waiting state, deadlock has occurred. Therefore, system is in unsafe state.

Date: _____________ **Signature of faculty in-charge**