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

Batch: B2	**Roll No.: 16010121194**
Experiment No. 07	
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)

The data structure I primarily used is Array (or List in python). The arrays used are:

- **`max_allocation`:** Used for storing the values of the maximum number of resources instances required by each process
- **`available`**: Used for storing the number of available instances of each resource
- **`need`**: Used for storing the number of resource instances still needed by the process
- **`allocated_process`:** Used for storing the number of resource instances already allocated to the process
- **`finish`**: Used to store the Boolean value signifying the completion of a process execution
- **`order`**: Used to store the order in which processes are executed

<u>Implementation details:</u> (printout of code)

```
num_instances = []
max_allocation = []
available = []
need = []
allocated_process = []
finish = []
order = []

num_processes = int(input("Enter the number of processes in the system: "))
num_resources = int(input("Enter the number of resources in the system: "))

print()
for i in range(num_resources):
    num = int(input("Enter the available number of instances for resource {}: ".format(i + 1)))
    available.append(num)

for i in range(num_processes):
```

```
    max_allocation_process = []
    print()
    for j in range(num_resources):
        maximum = int(input("Enter the max number of instances of 
resource {} needed by process {}: ".format(j + 1, i + 1)))
        max_allocation_process.append(maximum)

    max_allocation.append(max_allocation_process)

    allocated = []
    print()
    for j in range(num_resources):
        num_allocated = int(input("Enter the number of resource {} 
allocated to process {}: ".format(j + 1, i + 1)))
        allocated.append(num_allocated)

    allocated_process.append(allocated)


for i in range(num_processes):

    need_process = []
    finish.append(False)

    for j in range(num_resources):

        num = max_allocation[i][j] - allocated_process[i][j]
        need_process.append(num)

    need.append(need_process)

for index in range (0, 3):

    for i in range(num_processes):

        flag = 0
        temp = []

        if finish[i] != True:
            for j in range(num_resources):
```

```
                if need[i][j] <= available[j]:
                    tempvar = need[i][j] + available[j]
                    temp.append(tempvar)
                    flag += 1

                    if flag == 3:
                        for k in range(0, 3):
                            available[k] = temp[k]
                        finish[i] = True
                        order.append(i + 1)

flag2 = 1
for i in range(num_processes):
    if finish[i] == False:
        flag2 = 0
        print("The System is Unsafe")
        break
    else:
        continue

if(flag2 == 1):
    print("The System is Safe")
    print("The order is: ", order)
```

Ouput:

```
PS C:\Users\uditi> & C:/Users/uditi/AppData/Local/Programs/Python/Python3
omework/OS/Codes/EXP5_OS.py"
Enter the number of processes in the system: 5
Enter the number of resources in the system: 3

Enter the available number of instances for resource 1: 3
Enter the available number of instances for resource 2: 3
Enter the available number of instances for resource 3: 2

Enter the max number of instances of resource 1 needed by process 1: 7
Enter the max number of instances of resource 2 needed by process 1: 5
Enter the max number of instances of resource 3 needed by process 1: 3

Enter the number of resource 1 allocated to process 1: 0
Enter the number of resource 2 allocated to process 1: 1
Enter the number of resource 3 allocated to process 1: 0

Enter the max number of instances of resource 1 needed by process 2: 3
Enter the max number of instances of resource 2 needed by process 2: 2
Enter the max number of instances of resource 3 needed by process 2: 2

Enter the number of resource 1 allocated to process 2: 2
Enter the number of resource 2 allocated to process 2: 0
Enter the number of resource 3 allocated to process 2: 0

Enter the max number of instances of resource 1 needed by process 3: 9
Enter the max number of instances of resource 2 needed by process 3: 0
Enter the max number of instances of resource 3 needed by process 3: 2
```

```
Enter the max number of instances of resource 1 needed by process 3: 9
Enter the max number of instances of resource 2 needed by process 3: 0
Enter the max number of instances of resource 3 needed by process 3: 2

Enter the number of resource 1 allocated to process 3: 3
Enter the number of resource 2 allocated to process 3: 0
Enter the number of resource 3 allocated to process 3: 2

Enter the max number of instances of resource 1 needed by process 4: 2
Enter the max number of instances of resource 2 needed by process 4: 2
Enter the max number of instances of resource 3 needed by process 4: 2

Enter the number of resource 1 allocated to process 4: 2
Enter the number of resource 2 allocated to process 4: 1
Enter the number of resource 3 allocated to process 4: 1

Enter the max number of instances of resource 1 needed by process 5: 4
Enter the max number of instances of resource 2 needed by process 5: 3
Enter the max number of instances of resource 3 needed by process 5: 3

Enter the number of resource 1 allocated to process 5: 0
Enter the number of resource 2 allocated to process 5: 00
Enter the number of resource 3 allocated to process 5: 2
The System is Safe
The order is:  [2, 4, 5, 1, 3]
PS C:\Users\uditi>
```

Conclusion:

In this lab I understood and implemented Banker's algorithm which is used for deadlock avoidance using python. I used the "Need" matrix and compared its values with the number of resources available, and according manipulated resource instances.

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

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

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

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

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.

P4 needs 25 units, So, it will hold 35 units.

Process	Max	Hold	Need
P1	70	45	25
P2	60	40	20

P3	60	15	45
P4	60	35	25

Available = 150 - 45 - 40 - 15 - 35 = 15 units which are less than expected.
Thus, Deadlock occurs.

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

P4 max needs 60 units, initial need is 25 units

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

Available = 150 - 45 - 40 - 15 - 25 = 25 units which are enough to meet requirements.

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