**Batch: A2 Roll No.: 16010122041**

**Experiment No. 08**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:** Implementation of Banker’s Algorithm for Deadlock Avoidance Policy. |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**AIM:** Implementation of Process synchronization algorithms using mutexes and semaphore – Dining Philosopher problem

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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. **Sumitabha Das “ UNIX Concepts & Applications”, McGraw Hill Second**

**Edition.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Knowledge of deadlocks and all deadlock avoidance methods.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Description of the application to be implemented**:

The Banker's algorithm is a [resource allocation](http://en.wikipedia.org/wiki/Resource_allocation) and [deadlock](http://en.wikipedia.org/wiki/Deadlock) avoidance [algorithm](http://en.wikipedia.org/wiki/Algorithm) developed by [Edsger Dijkstra.](http://en.wikipedia.org/wiki/Edsger_Dijkstra)

# DATA STRUCTURES

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

**Implementation details:**

**CODE:**

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\_need = []

print("\nEnter max matrix: ")

for i in range(n):

temp = list(map(int, input().split()))

max\_need.append(temp)

# Set available resources directly

#avail = [0, 0, 0] # modify these values for testing

avail = [3, 3, 3]

f = [0] \* n

ans = [0] \* n

ind = 0

# Calculate need matrix

need = [[max\_need[i][j] - alloc[i][j] for j in range(m)] for i in range(n)]

# Banker's Algorithm logic

for k in range(n):

for i in range(n):

if f[i] == 0: # Process not finished

flag = 0

for j in range(m):

if need[i][j] > avail[j]: # Check if resources are available

flag = 1

break

if flag == 0: # Resources can be allocated

ans[ind] = i

ind += 1

for y in range(m):

avail[y] += alloc[i][y] # Release allocated resources

f[i] = 1 # Mark process as finished

# Output the safe sequence

if ind == n: # Check if all processes are finished

print("\nFollowing is the SAFE Sequence:")

for i in range(n - 1):

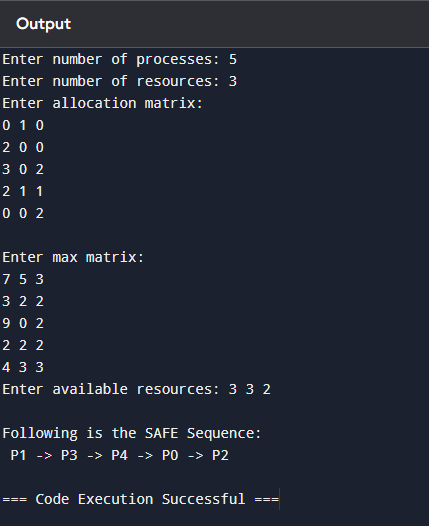
print(" P", ans[i], " ->", sep="", end="")

print(" P", ans[n - 1], sep="")

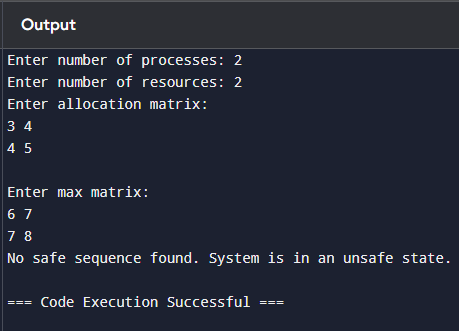
else:

print("No safe sequence found. System is in an unsafe state.")

**OUTPUT (NO DEADLOCK):**



**OUTPUT (DEADLOCK):**

****

**Conclusion:**

Banker’s Algorithm performed successfully.

**Post Lab Descriptive Questions**

1. **Explain the concept of a “safe state” in the context of the Banker’s Algorithm. Why is it important for deadlock avoidance?**

A "safe state" is a condition in which the system can allocate resources to each process in a way that ensures all processes can complete their execution without leading to a deadlock. In a safe state, there exists at least one sequence of process execution that allows all processes to finish.   
Identifying and maintaining a safe state is crucial for deadlock avoidance because it ensures that resource allocation decisions do not lead the system into a situation where no processes can proceed, thus preventing deadlocks from occurring.

1. **Describe the role of the need matrix in the Banker’s Algorithm. How is it calculated and used?**

**Need Matrix:**  
The need matrix represents the remaining resource needs of each process in the system. It is used to determine if a resource request can be granted without leading to an unsafe state. It helps in assessing whether the available resources are sufficient to meet the remaining needs of the processes, thereby guiding the system to maintain a safe state.

1. **What are the differences between deadlock prevention, avoidance, and detection? Provide examples of each.**

**Deadlock Prevention:**

* **Definition:** Strategies that ensure at least one of the necessary conditions for deadlock cannot hold.
* **Example:** Using mutual exclusion by ensuring that at least one resource is available (like allowing only one process to hold a lock).

**Deadlock Avoidance:**

* **Definition:** Dynamically checking resource requests and ensuring that the system remains in a safe state.
* **Example:** The Banker’s Algorithm checks if a resource request will lead to an unsafe state before granting it.

**Deadlock Detection:**

* **Definition:** The system allows deadlocks to occur but has mechanisms to detect and recover from them.
* **Example:** Using a resource allocation graph (RAG) to identify cycles that indicate deadlock

1. **How does the Resource Allocation Graph (RAG) help in detecting potential deadlocks?**

**Role of RAG in Deadlock Detection:**  
A Resource Allocation Graph visually represents the allocation of resources to processes. In this graph:

* Nodes represent processes and resources.
* Edges indicate resource allocation and request.
* If there is a cycle in the graph, it indicates a deadlock, as processes are waiting on each other in a circular manner. The presence of such a cycle triggers further investigation or recovery procedures.

1. **In the Banker’s Algorithm, what steps are taken if a resource request leads to an unsafe state?**

**Request Denial:** If a resource request leads to an unsafe state, the system denies the request.

**No Resource Allocation:** The process must wait until resources are available in a way that keeps the system in a safe state.

**Re-evaluation:** The system periodically re-evaluates pending requests to see if they can be granted without compromising safety.

1. **Analyze the impact of resource allocation policies on system throughput and process starvation. How can these policies be optimized?**

**Impact on Throughput and Starvation:**

* **Throughput:** Efficient resource allocation can maximize the number of processes completed in a given time frame. Poor policies can lead to bottlenecks.
* **Starvation:** Certain processes may wait indefinitely for resources if allocation policies favor others consistently.

**Optimization Strategies:**

* Implementing fair scheduling algorithms to balance resource allocation.
* Using priority levels to ensure that critical processes are not starved while still allowing for efficient throughput.

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

|  |  |  |
| --- | --- | --- |
| **Process** | **Max** | **Hold** |
| P1 | 70 | 45 |
| P2 | 60 | 40 |
| P3 | 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 P4 process arrives with max need of 60 and initial need of 25 units.

**Request from P4: Max Need = 60, Initial Need = 25**

**Request:** P4 requests 25 units.

* **Available before request:** 50 units
* **New Available after request:** 50 - 25 = 25 units
* **New Need for P4:** 60 - 25 = 35 units

**Updated State:**

* P1: Need = 25
* P2: Need = 20
* P3: Need = 45
* P4: Need = 35

**Safety Check:**

1. P1 can finish (Available = 25). New Available = 70.
2. P2 can finish (Available = 70). New Available = 110.
3. P3 can finish (Available = 110).

**Safe Sequence:** P1 → P2 → P3 → P4.  
**Result:** **Safe state.**

1. The P4 process arrives with max need of 60 and initial need of 35 units.

**Request from P4: Max Need = 60, Initial Need = 35**

**Request:** P4 requests 35 units.

* **Available before request:** 50 units
* **New Available after request:** 50 - 35 = 15 units
* **New Need for P4:** 60 - 35 = 25 units

**Updated State:**

* P1: Need = 25
* P2: Need = 20
* P3: Need = 45
* P4: Need = 25

**Safety Check:**

1. P1 cannot finish (Need = 25, Available = 15).
2. P2 cannot finish (Need = 20, Available = 15).
3. P3 cannot finish (Need = 45, Available = 15).

**Result:** **Not a safe state.**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**