**Batch: A-3 Roll No.: 16010122104**

**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.**

1. **William Stallings, “Operating System Internal & Design Principles”, Pearson.**
2. **Andrew S. Tanenbaum, “Modern Operating System”, Prentice Hall.**

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

**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 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) **Available:**

* 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 **Rj 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 **Pi** may request at most **‘k’** instances of resource type **Rj.**

**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 **Pi** is currently allocated **‘k’** instances of resource type **Rj 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 **Pi** currently need **‘k’** instances of resource type **Rj**
* **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: "))

# Input allocation matrix

alloc = []

print("Enter allocation matrix: ")

for i in range(n):

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

    alloc.append(temp)

# Input max matrix

max\_matrix = []

print("\nEnter max matrix: ")

for i in range(n):

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

    max\_matrix.append(temp)

# Input available resources

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

# Initialize

f = [0] \* n

ans = [0] \* n

ind = 0

# Calculate need matrix

need = [[0 for i in range(m)] for i in range(n)]

for i in range(n):

    for j in range(m):

        need[i][j] = max\_matrix[i][j] - alloc[i][j]

# Safety algorithm

for k in range(n):

    f[k] = 0

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

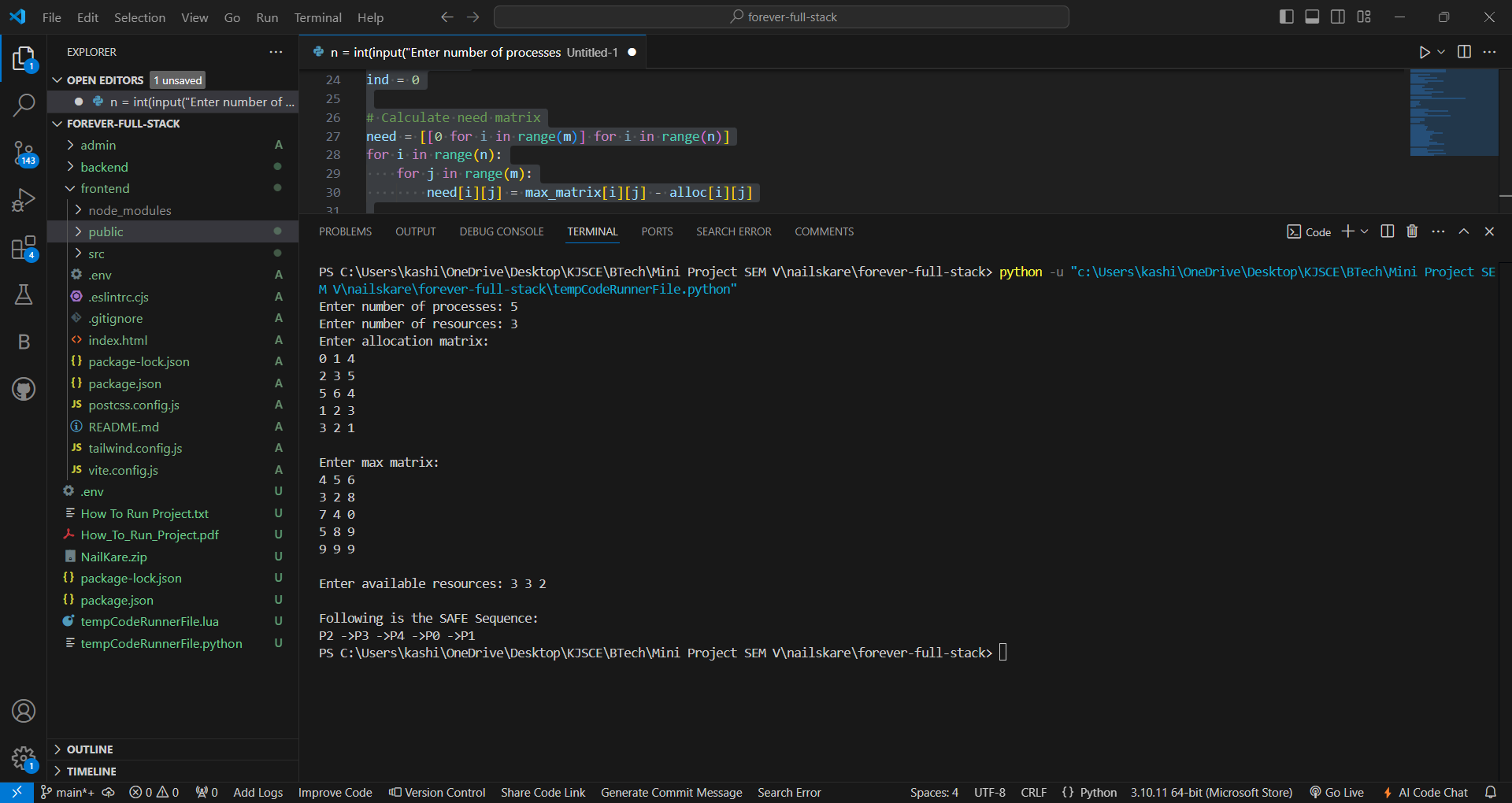
# Print safe sequence

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

for i in range(n - 1):

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

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

****

**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:
   1. All resources have a single instance
   2. All resources have multiple instances
   3. Both a and b
   4. None of the above

**Ans:**

1. Resources are allocated to the process on non-sharable basis is \_
   1. Hold and Wait
   2. Mutual Exclusion
   3. No pre-emption
   4. Circular Wait

**Ans:**

1. Which of the following approaches require knowledge of the system state?
   1. Deadlock Detection
   2. Deadlock Prevention
   3. Deadlock Avoidance
   4. All of the above

**Ans:**

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

