

**Experiment-7**

Operating Systems

**Program:**

MCA

**SVKM'S-NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**School of Technology Management and Engineering**

**[2024-25]**

Lab Manual

PART B

***(Students must submit the soft copy as per following segments as per the submission instructions.)***

| Roll No.: A073 | Name: Aryan Srivastava |
| --- | --- |
| Class: MCA SEM1 | Batch: B3 |
| Experiment Number- | |
| Date of Experiment: | Date of Submission: |
| Grade: |  |

**B.1 Program with Output to be written by student**

def is\_safe(processes, available, max\_demand, allocation, need):

    n = len(processes)

    m = len(available)

    work = available[:]

    finish = [False] \* n

    safe\_sequence = []

    while len(safe\_sequence) < n:

        found\_process = False

        for i in range(n):

            if not finish[i]:

                if all(need[i][j] <= work[j] for j in range(m)):

                    for j in range(m):

                        work[j] += allocation[i][j]

                    finish[i] = True

                    found\_process = True

                    safe\_sequence.append(processes[i])

                    break

        if not found\_process:

            return False, []

    return True, safe\_sequence

def bankers\_algorithm():

    processes = ['P1', 'P2', 'P3', 'P4', 'P5']

    resource\_types = ['A', 'B', 'C']

    available = [3, 3, 2]

    max\_demand = [

        [7, 5, 3],

        [3, 2, 2],

        [9, 0, 2],

        [2, 2, 2],

        [4, 3, 3]

    ]

    allocation = [

        [0, 1, 0],

        [2, 0, 0],

        [3, 0, 2],

        [2, 1, 1],

        [0, 0, 2]

    ]

    need = [[max\_demand[i][j] - allocation[i][j] for j in range(3)] for i in range(5)]

    safe, safe\_sequence = is\_safe(processes, available, max\_demand, allocation, need)

    if safe:

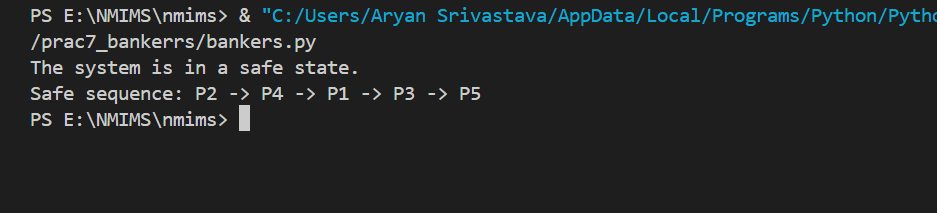
        print("The system is in a safe state.")

        print("Safe sequence:", " -> ".join(safe\_sequence))

    else:

        print("The system is not in a safe state.")

bankers\_algorithm()



**B.2 Answers the following question based on study**

**Example:** Consider a system that contains five processes P1, P2, P3, P4, P5 and the three resource types A, B and C. Following are the resources types: A has 10, B has 5 and the resource type C has 7 instances.

|  |  |  |  |
| --- | --- | --- | --- |
| **Process** | **Allocation A         B         C** | **Max A         B         C** | **Available A         B         C** |
| P1 | 0         1          0 | 7         5         3 | 5         3         2 |
| P2 | 2         0         0 | 3         2         2 |  |
| P3 | 3         0         2 | 9         0         2 |  |
| P4 | 2         1         1 | 2         2         2 |  |
| P5 | 0         0         2 | 3         2         3 |  |

**Answer the following questions using the banker's algorithm:**

1. What is the reference of the need matrix?

The Need matrix represents the remaining resources that each process requires to complete its execution.

1. Determine if the system is safe or not.

To determine whether the system is in a safe state, we apply the Banker's algorithm. The available resources are:

Available=[3,3,2]\text{Available} = [3, 3, 2]Available=[3,3,2]

We now use this available vector and the Need matrix to check if there is a sequence of processes that can finish one by one without causing a deadlock.

Step-by-step Execution:

Start with available resources [3, 3, 2].

Look for a process where the Need can be satisfied by the available resources.

P1 needs [7, 4, 3], which cannot be satisfied by [3, 3, 2]. Skip.

P2 needs [1, 2, 2], which can be satisfied by [3, 3, 2]. So, allocate the resources to P2, and after P2 finishes, the available resources will be [3 + 2, 3 + 0, 2 + 0] = [5, 3, 2].

P3 needs [6, 0, 0], which cannot be satisfied by [5, 3, 2]. Skip.

P4 needs [0, 1, 1], which can be satisfied by [5, 3, 2]. So, allocate the resources to P4, and after P4 finishes, the available resources will be [5 + 2, 3 + 1, 2 + 1] = [7, 4, 3].

P5 needs [4, 3, 1], which can be satisfied by [7, 4, 3]. So, allocate the resources to P5, and after P5 finishes, the available resources will be [7 + 0, 4 + 0, 3 + 2] = [7, 4, 5].

Now P1 needs [7, 4, 3], which can be satisfied by [7, 4, 5]. Allocate the resources to P1, and after P1 finishes, the available resources will be [7 + 0, 4 + 1, 5 + 0] = [7, 5, 5].

Finally, P3 needs [6, 0, 0], which can be satisfied by [7, 5, 5]. So, allocate the resources to P3, and after P3 finishes, the available resources will be [7 + 3, 5 + 0, 5 + 2] = [10, 5, 7].

Since we were able to find a sequence in which all processes can finish, the system is in a safe state.

Safe Sequence: P2 → P4 → P5 → P1 → P3

1. What will happen if the resource request (1, 0, 0) for process P1 can the system accept this request immediately?

To check if the system can accept the resource request (1, 0, 0) from P1, we need to evaluate if the requested resources are less than or equal to both P1's Need and the system's Available resources.

P1's Need is [7, 4, 3].

Available resources are [3, 3, 2].

The request is [1, 0, 0].

Since the request is less than or equal to both the Need ([7, 4, 3]) and Available ([3, 3, 2]), we can grant the request temporarily and check if the system remains in a safe state.

After the request:

The Available resources would become [3 - 1, 3 - 0, 2 - 0] = [2, 3, 2].

P1's Allocation would become [0 + 1, 1 + 0, 0 + 0] = [1, 1, 0].

P1's Need would become [7 - 1, 4 - 0, 3 - 0] = [6, 4, 3].

Now, we check if the system is in a safe state with these updated values:

New Available resources: [2, 3, 2]

New Allocation for P1: [1, 1, 0]

New Need for P1: [6, 4, 3]

Repeating the safety check process from above, the system remains in a safe state with the same safe sequence: P2 → P4 → P5 → P1 → P3.

Therefore, the system can accept the request (1, 0, 0) for P1 immediately without entering an unsafe state.