Northern University
Of Business and Technology Khulna

Lab report-03

Course Code: CSE-3108

Course Title: Operating System Lab

Submitted by:

Name : Nazmus Sakib
Section : 5B
ID : 11220320888
Department: Computer Science and Engineering

Submitted to:

Name : Md. Tahmid Hasan
Designation: Coordinator & Senior Lecturer
Department: Computer science and engineering

Remarks

Date of Submission: 18.03.2025

Problem: 03

Problem name: Banker's Algorithm

Objectives: To implement the Banker’s Algorithm, which ensures safe resource allocation by checking whether granting a request keeps the system in a safe state. This algorithm prevents deadlocks by ensuring that resources are allocated only if they do not lead to an unsafe state.

Algorithm:

1. Input the following:
   - Allocation matrix (resources already allocated).
   - Max matrix (maximum demand of each process).
   - Available array (available resources).
2. Compute the Need matrix as the difference between the Max matrix and the Allocation matrix: Need[i][j]=Max[i][j]−Allocation[i][j]
3. Initialize a Finish array with False for all processes.
4. Check for a safe sequence:
   - Find a process P[i] whose Need[i] is less than or equal to Available.
   - If found, allocate resources temporarily and update Available.
   - Mark P[i] as finished and add it to the safe sequence.
   - Repeat until all processes are completed or no valid process is found.
5. If all processes finish successfully, the system is in a safe state; otherwise, it is in an unsafe state.

Code:

```
def is_safe_state(processes, allocation, max_demand, available):
    n = len(processes)
    m = len(available)

    # Compute Need matrix
    need = [[max_demand[i][j] - allocation[i][j] for j in range(m)] for i in range(n)]

    # Initialize work and finish arrays
    work = available[:]
    finish = [False] * n
    safe_sequence = []

    # Find a safe sequence
    while len(safe_sequence) < n:
        allocated = False
        for i in range(n):
            if not finish[i] and all(need[i][j] <= work[j] for j in range(m)):
                for j in range(m):
                    work[j] += allocation[i][j]
```

```
                safe_sequence.append(processes[i])
                finish[i] = True
                allocated = True
                break
        if not allocated:
            return False, []

    return True, safe_sequence

# Example input
processes = [0, 1, 2, 3, 4]
allocation = [
    [0, 1, 0],
    [2, 0, 0],
    [3, 0, 2],
    [2, 1, 1],
    [0, 0, 2]
]
max_demand = [
    [7, 5, 3],
    [3, 2, 2],
    [9, 0, 2],
    [2, 2, 2],
    [4, 3, 3]
]
available = [3, 3, 2]

# Run the Banker's Algorithm
safe, safe_sequence = is_safe_state(processes, allocation, max_demand, available)

if safe:
    print("System is in a safe state.")
    print("Safe Sequence:", safe_sequence)
else:
    print("System is in an unsafe state. Deadlock possible.")
```

Output:

```
System is in a safe state.
Safe Sequence: [1, 3, 0, 2, 4]
PS E:\OS LAB FINAL>
```

Conclusion: The Banker's Algorithm ensures that resources are allocated safely without causing a deadlock. If a safe sequence exists, the system remains stable; otherwise, it enters an unsafe state, potentially leading to a deadlock. However, the algorithm requires accurate prior knowledge of resource demands, making it more theoretical than practical in dynamic environments.