LAB Manual

Name of Student: Vidhi Binwal	**PRN 22070122249**
Semester: IV	**Year AY 23-24**
Subject Title: Operating Systems Lab	
EXPERIMENT No: 12	**Assignment No : 9**
TITLE: Banker's algorithm	**DoP : 22/4/24**

Aim: Implement C program demonstrate **Banker's algorithm**

Learning Outcomes: *1. To understand the* **deadlock concept**

2. To Demonstrate the **Banker's** algorithm

***Hardware/Software*:**

* HARDWARE / SOFTWARE :- PC, Dev C++, processor with 1.6 GHz clock speed or fa[ster]

Theory:

* THEORY :-

The Banker's algorithm is a resource allocation and deadlock avoidance method in operating systems.

It ensures that resources are allocated to processes in a way that prevents deadlock. By considering the current allocation and future resource requests, it maintains system integrity and efficient resource usage.

Algorithm:

1. Initialize:

   - Available: Vector representing the number of available instances of each resource.

   - Max: Matrix indicating the maximum demand of each process.

   - Allocation: Matrix showing the resources currently allocated to each process.

   - Need: Matrix representing the remaining resources needed by each process (Max - Allocation).

2. When a process requests resources:

   a. Check if the requested resources are less than or equal to the available resources.

   b. Check if the requested resources are less than or equal to the remaining need of the process.

   c. If both conditions are met, pretend to allocate the resources to the process and perform a safety check (explained below).

   d. If the safety check passes, allocate the resources; otherwise, deny the request.

3. Safety Check:

   a. Maintain two vectors: Work (copies Available) and Finish (initially all false).

   b. Find a process that can complete its execution by comparing its Need with the Work vector. If found, pretend to allocate its resources and mark it as finished.

   c. Repeat step b until all processes can complete. If all processes can finish, the system is in a safe state; otherwise, it's unsafe.

4. If a request is granted, update the allocation and available vectors. If denied, the system remains unchanged.

Program:

```cpp
#include <iostream>
#include <vector>
using namespace std;

const int P = 3;
const int R = 2;

bool isSafe(int available[], int max[][R], int allocated[][R], int need[][R], int process) {
}

int main() {

    int available[R] = {2, 1};

    int max[P][R] = { {5, 4},
                      {2, 1},
                      {3, 2} };

    int allocated[P][R] = { {1, 2},
                            {1, 0},
                            {1, 1} };

    int need[P][R];
    cout << "Need Matrix:\n";
    for (int i = 0; i < P; i++) {
        for (int j = 0; j < R; j++) {
            need[i][j] = max[i][j] - allocated[i][j];
            cout << need[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;

    if (isSafe(available, max, allocated, need, 0))
        cout << "System is in safe state\n";
    else
        cout << "System is in unsafe state\n";

    return 0;
}
```

Output:

```
Need Matrix:
4 2
1 1
2 1

System is in safe state

--------------------------------
Process exited after 0.2678 seconds with return value 0
Press any key to continue . . .
```

Conclusion:

* CONCLUSION:- Hence, we studied the Banker's algo. and can now successfully implement it to prevent or avoid deadlock situations in Operating Systems.