Experiment No. 6

Aim: To study & implementation of Banker's Algorithm for deadlock avoidance.

Objective: To understand & implement the Banker's Algo. used for deadlock avoidance in ~~an~~ operating system.

Aim: To study & implementation of Banker's Algorithm for deadlock Avoidance

Objective: To understand & implement the Banker's Algo used for deadlock avoidance in an operating system.

About Deadlocks: A deadlock is an situation in an OS where a set of processers are blocked because each of a process is holding a resource & waiting for another resource acquired by some other processes.

About Banker's Algorithm: Banker's Algorithm is a deadlock avoidance algorithm that checks the system's safety state before allocating requested resources to ensure that the system does not enter a deadlock state. It was developed by Edsger ~~Dijsk~~ Dijkstra & is called "Banker's" because it is similar to how a banker allocates cash to clients ensuring solvency.

Key Components:

i) Available : A vector that indicates that the number of the available instances of each resource type.

ii) Max : A matrix that defines the maximum demand of each process.

iii) Allocation : A matrix that shows the number of resources currently allocated to each process.

iv) Need : A matrix calculated as

$$Need[i][j] = Max[i][j] - Allocation[i][j]$$

Algorithm Steps :

i) Calculate the Need matrix

ii) Check if the system is in safe state :

- Find a process whose needs can be satisfied with the current available resources.
- Assume the process finishes & releases it's resources
- Repeat until all processes can finish or no such process can be found.

iii) If all processes can finish, the system is in a safe state.

iv) Otherwise, the system is in an unsafe state & may lead to deadlock.

Algorithm Working :

i) Input the number of processes & resource types.

ii) Input the allocation, Max & Available Matrices.

iii) Compute the need matrix.

iv) Apply Banker's algorithm to determine weather the system is in a safe mode / state.

v) Display the safe sequence if one exists

Code implementation :

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

const int P = 5;
const int R = 3;

int main () {
    int allocation [P][R] = { {0, 1, 0}, {2,0,0},
    {3,0,2}, {2,1,1}, {0,0,2}};

    int max [P][R] = { {7,5,3}, {3,2,2},
    {9,0,2}, {2, 2, 2}, {4,3,3}};

    int available [R] = {3,3,2};

    int need [P][R];
    for (int i=0; i<P; i++)
        for (int j=0; j<R; j++)
            need [i][j] = max [i][j] -
                allocation [i][j];
    bool finish [P] = {0};
    int safeSeq [P];
```

```
int work[R];
for(int i=0; i< R; i++)
        work[i] = avaliable [i];

int count = 0;
while (count < P) {
    bool found = false;
    for (int p=0; p < P; p++) {
        if(!finish [p]) {
            bool canAllocate = true;
            for (int r = 0; r <R; r++) {
                if(need [p][r] > work[r]) {
                    canAllocate = false;
                    break;
                }
            }
    if(canAllocate) {
        for (int k=0; k<R; k++)
            work[k] += allocation [p][k];

        safeSeq [count++] = p;
        finish[p] = true;
        found = true;
    }
  }
}
```

Conclusion: The banker's Algorithm is a vital technique for deadlock avoidance in operating systems. It ensures that the system always remains in a safe mode by simulating allocation & checking feasibility before actual resource allocation.

9/4/25


```
if (!found) {
        cout << "System is NOT in safe state"
                  << endl;
        return 0;
    }
}
cout << "System is in safe state. In Safe Sequence
          is : ";
for (int i = 0 ; i < P ; i++) {
    cout << "P" << safeSeq [i] << (i == P-1
            ? "\n" : "-->");
    return 0;
}
```

Output : System is in a safe state
Safe sequence is : P1 -> P3 -> P4 -> P0 -> P2

Conclusion: The banker's algorithm is a vital technique for deadlock avoidance in operating systems. It ensures that the system always remains in a safe mode by simulating allocation & checking feasibility before actual resource allocation

9/4/25