Laboratory 6

Title of the Laboratory Exercise: Programs for deadlock avoidance algorithm

1. Introduction and Purpose of Experiment

   Deadlocks can be avoided if certain information is available in advance. By solving these problems students will become familiar to avoid deadlock in advance with the available resource information

2. Aim and Objectives

   Aim

   - To develop Bankers algorithm for multiple resources for deadlock avoidance

   Objectives

   At the end of this lab, the student will be able to

   - Verify a problem to check that whether deadlock will happen or not for the given resources
   - Implement the bankers algorithm for multiple resources

3. Experimental Procedure

   i. Analyse the problem statement

   ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code

   iii. Implement the algorithm in C language

   iv. Compile the C program

   v. Test the implemented program

   vi. Document the Results

   vii. Analyse and discuss the outcomes of your experiment

4. Questions

   Implement a Bankers algorithm for deadlock avoidance

5. Calculations/Computations/Algorithms

Safety Algorithm

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.

Initialize: Work = Available

Finish[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both

a) Finish[i] = false

b) Needi <= Work

if no such i exists goto step (4)

3) Work = Work + Allocation[i]

Finish[i] = true

goto step (2)

4) if Finish [i] = true for all i

then the system is in a safe state

Resource-Request Algorithm

1) If Requesti <= Needi

Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If Requesti <= Available

Goto step (3); otherwise, Pi must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

Available = Available – Requesti

Allocationi = Allocationi + Requesti

Needi = Needi- Requesti

6. Presentation of Results

```cpp
#include <iostream>
#include <vector>

int main(int, char**) {
    int n = 5;  // number of processes
    int m = 3;  // number of resources

    using matrix = std::vector<std::vector<int>>;

    matrix alloc{{0, 1, 0},
                 {2, 0, 0},
                 {3, 0, 2},
                 {2, 1, 1},
                 {0, 0, 2}};

    matrix max{{7, 5, 3},
               {3, 2, 2},
               {9, 0, 2},
               {2, 2, 2},
               {4, 3, 3}};

    std::vector<int> avail{3, 3, 2};

    matrix need;

    for (int i = 0; i < n; i++) {
        std::vector<int> temp;
        for (int j = 0; j < m; j++) {
            temp.push_back(max.at(i).at(j) - alloc.at(i).at(j));
        }
        need.push_back(temp);
    }

    int y = 0;
    std::vector<int> f(n);
    std::vector<int> ans;

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            if (f.at(i) == 0) {
                int flag = 0;

                for (int j = 0; j < m; j++) {
                    if (need.at(i).at(j) > avail.at(j)) {
                        flag = 1;
                        break;
                    }
                }

                if (flag == 0) {
```

```cpp
                    ans.push_back(i);
                    for (int y = 0; y < m; y++)
                        avail.at(y) += alloc.at(i).at(y);
                    f.at(i) = 1;
                }
            }
        }
    }

    std::cout << "The Safe Sequence is : " << std::endl;
    for (auto a : ans)
        std::cout << "P" << a << " -> ";
    std::cout << "END" << std::endl;
}
```
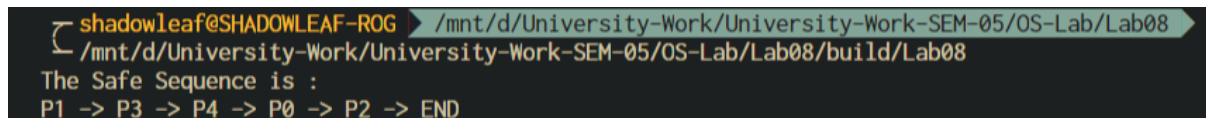
7. Analysis and Discussions


```
shadowleaf@SHADOWLEAF-ROG  /mnt/d/University-Work/University-Work-SEM-05/OS-Lab/Lab08
  /mnt/d/University-Work/University-Work-SEM-05/OS-Lab/Lab08/build/Lab08
The Safe Sequence is :
P1 -> P3 -> P4 -> P0 -> P2 -> END
```

8. Conclusions

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

In simple terms, it checks if allocation of any resource will lead to deadlock or not, OR is it safe to allocate a resource to a process and if not then resource is not allocated to that process. Determining a safe sequence(even if there is only 1) will assure that system will not go into deadlock.

Banker's algorithm is generally used to find if a safe sequence exist or not.

Consider there are n account holders in a bank and the sum of the money in all of their accounts is S. Every time a loan has to be granted by the bank, it subtracts the loan amount from the total money the bank has. Then it checks if that difference is greater than S. It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

Banker's algorithm works in a similar way in computers. Whenever a new process is created, it must exactly specify the maximum instances of each resource type that it needs. The banker's algorithm is a method used in deadlock avoidance technique in multiple instances of a resource type.

9. Comments

1. Limitations of Experiments

- It requires the number of processes to be fixed; no additional processes can start while it is executing.

- It allows all requests to be granted in finite time, but one year is a finite amount of time.

- Similarly, all of the processes guarantee that the resources loaned to them will be repaid in a finite amount of time. While this prevents absolute starvation, some pretty hungry processes might develop.

2. Limitations of Results

All processes must know and state their maximum resource need in advance.

It requires that the number of resources remain fixed; no resource may go down for any reason without the possibility of deadlock occurring.

3. Learning happened

We learnt how to use the Banker's algorithm for deadlock avoidance and determining a safe path.

4. Recommendations

The bankers algorithm implemented here can be further improved to find all the possible safe paths, so that we can assign processes with higher priority the resources.