Name: K Srikanth Roll No: 17ETCS002124

Laboratory 7

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 banker's 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

Safe State

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

Finish[i] = false

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

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

C Program Code

```c
#include <stdio.h>
int main()
{
    // 17ETCS002124 K Srikanth
    int n, m, i, j, k;
    n = 5;
    m = 3;
    int alloc[5][3] = { { 0, 1, 0 },{ 2, 0, 0 }, { 3, 0, 2 }, { 2, 1, 1 }, { 0, 0, 2 } };
    int max[5][3] = { { 7, 5, 3 },  { 3, 2, 2 }, { 9, 0, 2 }, { 2, 2, 2 }, { 4, 3, 3 } };
    int avail[3] = { 3, 3, 2 };
    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++) {
        f[k] = 0; }
    int need[n][m];
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j]; }
    int y = 0;
    for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
            if (f[i] == 0) {
                int flag = 0;
                for (j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]){
                        flag = 1;
                        break; }}
                if (flag == 0) {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;}}}}
    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);
    return (0);

}
```

Figure 1 C Program for the given problem statement

Output

To Run this C Program

```
>> gcc -W -Wall filename.c -o filename -pthread
```

Now to run the executable file,

```
>> ./filename
```

Result

```
❯ gcc lab_7.c
❯ ./a.out
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2%
```

Figure 2 C Program output for the given problem statement

7. Analysis and Discussions

Let 'P' be the number of processes in the system and 'R' be the number of resources types, is predefined in the program as 3 and 5.

Maximum:

- It is a 2-d array of size 'P*R' that defines the maximum demand for each resource by each process in a system.
- Max[i, j] = k means process Pi may request at most 'k' instances of resource type Rj.

Allocation:

- It is a 2-d array of size 'P*R' that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process Pi is currently allocated 'k' instances of resource type Rj

Need:

- it is a 2-d array of size 'P*R' that indicates the remaining resource need of each process.
- Need [i, j] = k means process Pi currently allocated 'k' instances of resource type Rj
- Need [i, j] = Max [i, j] – Allocation [i, j]

Allocation i specifies the resources currently allocated to process Pi and Need i specifies the additional resources that process Pi may still request to complete its task. Considering a system with five processes P0 through P4 and three resources types A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Available instances are 3, 3 and 2. Find a process which can be completed using available resources, complete the process and free the allocated resources to the available resources

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.

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 requires that the number of resources remain fixed; no resource may go down for any reason without the possibility of deadlock occurring.
- All processes must know and state their maximum resource need in advance.

2. Limitations of Results

- 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.

3. Learning happened

The Banker's algorithm:
Allows: mutual exclusion, wait and hold and no pre-emption Prevents: circular wait.