

Assignment No 4:

Name: Samruddhi Devram Khilari

Branch & Year: AIML-B SY

Prn: 12420020

Roll no: 2

Sub: Operating System Lab

#Deadlock Handling Approach Algorithms. => avoids Deadlock into processes.

1. Q] Implement Banker's Algorithm.

The screenshot shows a dual-terminal setup. The left terminal window displays the source code for the Banker's algorithm in C, titled 'FCFS_process_scheduling.c'. The right terminal window displays the output of the program, titled '*bankers_alog.c'.

Code (FCFS_process_scheduling.c):

```
1 #include <stdio.h>
2 int main()
3 {
4     // P0, P1, P2, P3, P4 are the Process names here
5
6     int n, m, i, j, k;
7     n = 5;                                // Number of processes
8     m = 3;                                // Number of resources
9     int alloc[5][3] = {{0, 1, 0},           // P0 // Allocation Matrix
10        {2, 0, 0},                         // P1
11        {3, 0, 2},                         // P2
12        {2, 1, 1},                         // P3
13        {0, 0, 2}};                        // P4
14
15    int max[5][3] = {{7, 5, 3},            // P0 // MAX Matrix
16        {3, 2, 2},                         // P1
17        {9, 0, 2},                         // P2
18        {2, 2, 2},                         // P3
19        {4, 3, 3}};                        // P4
20
21    int avail[3] = {3, 3, 2};             // Available Resources
22
23    int f[n], ans[n], ind = 0;
24    for (k = 0; k < n; k++)
25    {
26        f[k] = 0;
27    }
28    int need[n][m];
29    for (i = 0; i < n; i++)
30    {
31        for (j = 0; j < m; j++)
32            need[i][j] = max[i][j] - alloc[i][j];
33    }
34
35    for (i = 0; i < n; i++)
36    {
37        for (j = 0; j < m; j++)
38            if (f[j] >= need[i][j])
39            {
40                f[j] -= need[i][j];
41                ans[i] = 1;
42            }
43        if (ans[i] == 1)
44        {
45            for (j = 0; j < m; j++)
46                if (f[j] <= 0)
47                    break;
48            if (j == m)
49            {
50                for (i = 0; i < n; i++)
51                    f[i] += alloc[i][j];
52                ind++;
53            }
54        }
55    }
56    if (ind == n)
57    {
58        printf("The following system is not safe");
59    }
60    else
61    {
62        printf("The following system is safe");
63    }
64
65    // DISPLAY
66    printf("      Max          Allocate        Need \n");
67    for (int i=0;i<n;i++) {
68        for(int j=0;j<m;j++) {
69            printf(" %d  %d  %d  ",max[i][j],alloc[i][j],need[i][j]);
70        }
71        printf("\n");
72    }
73    if (flag == 1){
74        printf("Following is the SAFE Sequence\n");
75        for (i = 0; i < n - 1; i++)
76            printf("%d ->", ans[i]);
77        printf("%d", ans[n - 1]);
78    }
79    return (0);
80 }
```

Output (*bankers_alog.c):

```
vbox@ubuntu:~/Desktop/OS$ gcc bankers_alog.c
vbox@ubuntu:~/Desktop/OS$ ./a.out
      Max          Allocate        Need
    7  0    7      5  1    4      3  0    3
    3  2    1      2  0    2      2  0    2
    9  3    6      0  0    0      2  2    0
    2  2    0      2  1    1      2  1    1
    4  0    4      3  0    3      3  2    1
Following is the SAFE Sequence
P1 -> P3 -> P4 -> P0 -> P2vbox@ubuntu:~/Desktop/OS$
```

Output ===

The screenshot shows a terminal window with the command 'vbox@ubuntu:~/Desktop/OS\$' at the prompt. The user runs 'gcc bankers_alog.c' and then './a.out'. The program outputs the allocation matrix, the maximum resource requirement matrix, and the need matrix. It then prints the safe sequence of processes: P1 -> P3 -> P4 -> P0 -> P2.

```
vbox@ubuntu:~/Desktop/OS$ gcc bankers_alog.c
vbox@ubuntu:~/Desktop/OS$ ./a.out
      Max          Allocate        Need
    7  0    7      5  1    4      3  0    3
    3  2    1      2  0    2      2  0    2
    9  3    6      0  0    0      2  2    0
    2  2    0      2  1    1      2  1    1
    4  0    4      3  0    3      3  2    1
Following is the SAFE Sequence
P1 -> P3 -> P4 -> P0 -> P2vbox@ubuntu:~/Desktop/OS$
```

Q] Deadlock Detection Algorithm

```
#include <stdio.h>
#include <stdbool.h>

#define P 5 // Number of processes
#define R 3 // Number of resources

int allocation[P][R]; // Allocation matrix
int max[P][R]; // Maximum matrix
int need[P][R]; // Need matrix
int available[R]; // Available resources

// Function to calculate the need matrix
void calculateNeed() {
    for (int i = 0; i < P; i++) {
        for (int j = 0; j < R; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}

// Function to detect deadlock using the wait-for graph
bool isDeadlocked() {
    bool finish[P] = {false};
    int count = 0;

    while (count < P) {
        bool found = false;

        for (int p = 0; p < P; p++) {
            // Check if process p can finish
            if (!finish[p]) {
                bool canFinish = true;
                for (int j = 0; j < R; j++) {
                    if (need[p][j] > available[j]) {
                        canFinish = false;
                        break;
                    }
                }
                if (canFinish) {
                    // Simulate finishing process p
                    for (int j = 0; j < R; j++) {
                        available[j] += allocation[p][j]; // Release allocated resources
                    }
                    finish[p] = true;
                    count++;
                    found = true;
                }
            }
        }
        if (!found) {
            // If no process could finish, we have a deadlock
            printf("Deadlock detected among the following processes:\n");
            for (int i = 0; i < P; i++) {
                if (!finish[i]) {

```

```

        printf("Process %d\n", i);
    }
}
return true;
}

printf("No deadlock detected.\n");
return false;
}

int main() {
    // Initialize allocation, maximum, and available resources
    int allocationInput[P][R] = {
        {0, 1, 0},
        {2, 0, 0},
        {3, 0, 2},
        {2, 1, 1},
        {0, 0, 2}
    };

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

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

    // Copy inputs to global matrices
    for (int i = 0; i < P; i++) {
        for (int j = 0; j < R; j++) {
            allocation[i][j] = allocationInput[i][j];
            max[i][j] = maxInput[i][j];
        }
    }

    for (int i = 0; i < R; i++) {
        available[i] = availableInput[i];
    }

    // Calculate the need matrix
    calculateNeed();

    // Check for deadlock
    isDeadlocked();

    return 0;
}

```

Output ===

No deadlock detected.