Lab No: 13 **Date: 2082/**

Title: Write a program to test if the system is free from dead lock or not for the user input allocation, max and available matrix.

Deadlock is a situation in a multiprogramming environment where a set of processes are blocked because each process is holding a resource and waiting for another resource held by some other process. In simpler terms, processes cannot proceed because they are waiting for resources that are held by other waiting processes, creating a cycle of dependencies. To ensure smooth execution of processes, it is essential to check whether the system is free from deadlock. The Deadlock Detection Algorithm is used to detect deadlocks in a system by analyzing the allocation, maximum demand, and availability of resources.

Algorithm:

Step 1: Initialize a Work vector as the same as the Available resources vector V.

Step 2: Initialize a Finish array of size equal to the number of processes and set all entries to false.

Step 3: Find a process P[i] such that Finish[i] = false and Need[i] <= Work.

- If such a process exists, add its allocated resources to Work (Work = Work + Allocation[i]) and set Finish[i] = true.
- Repeat Step 3 until no such process is found.

Step 4: If all Finish[i] = true, then the system is safe and free from deadlock.

- If some Finish[i] = false, then those processes are deadlocked.

Language: C++

IDE: VS Code

Code:

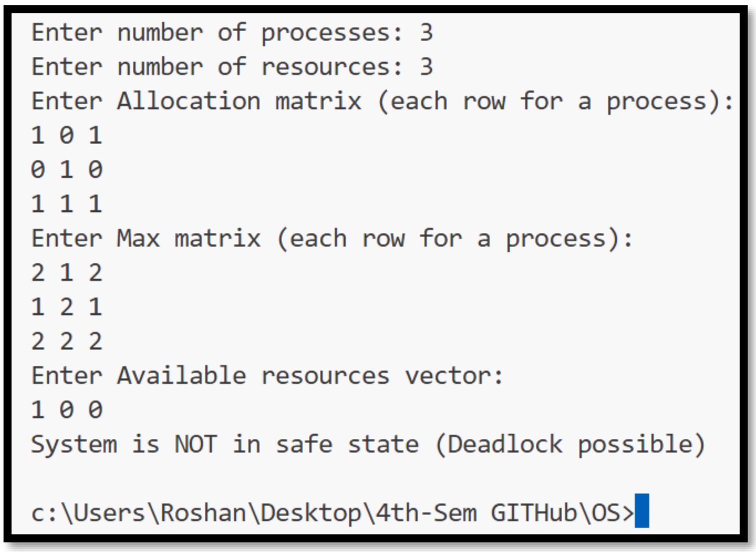
```
#include <stdio.h>

#include <stdbool.h>


int main() {

    int n, m;  // n = number of processes, m = number of resource types

    printf("Enter number of processes: ");

    scanf("%d", &n);

    printf("Enter number of resources: ");

    scanf("%d", &m);

    int allocation[n][m];

    int max[n][m];

    int available[m];

    int need[n][m];

    printf("Enter Allocation matrix (each row for a process):\n");

    for (int i = 0; i < n; i++)

        for (int j = 0; j < m; j++)

            scanf("%d", &allocation[i][j]);

    printf("Enter Max matrix (each row for a process):\n");

    for (int i = 0; i < n; i++)

        for (int j = 0; j < m; j++)

            scanf("%d", &max[i][j]);

    printf("Enter Available resources vector:\n");

    for (int i = 0; i < m; i++)

        scanf("%d", &available[i]);
```

```
// Calculate Need matrix
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        need[i][j] = max[i][j] - allocation[i][j];
bool finish[n];
for (int i = 0; i < n; i++)
    finish[i] = false;
int safeSeq[n];
int count = 0;
while (count < n) {
    bool found = false;
    for (int p = 0; p < n; p++) {
        if (!finish[p]) {
            bool canAllocate = true;
            for (int r = 0; r < m; r++) {
                if (need[p][r] > available[r]) {
                    canAllocate = false;
                    break;
                }
            }
            if (canAllocate) {
                for (int r = 0; r < m; r++)
                    available[r] += allocation[p][r];
                safeSeq[count++] = p;
                finish[p] = true;
```

```
                    found = true;
                }
            }
        }
        if (!found) {
            printf("System is NOT in safe state (Deadlock possible)\n");
            return 0;
        }
    }
    printf("System is in safe state.\nSafe sequence is: ");
    for (int i = 0; i < n; i++)
        printf("P%d ", safeSeq[i]);
    printf("\n");
    return 0;
}
```

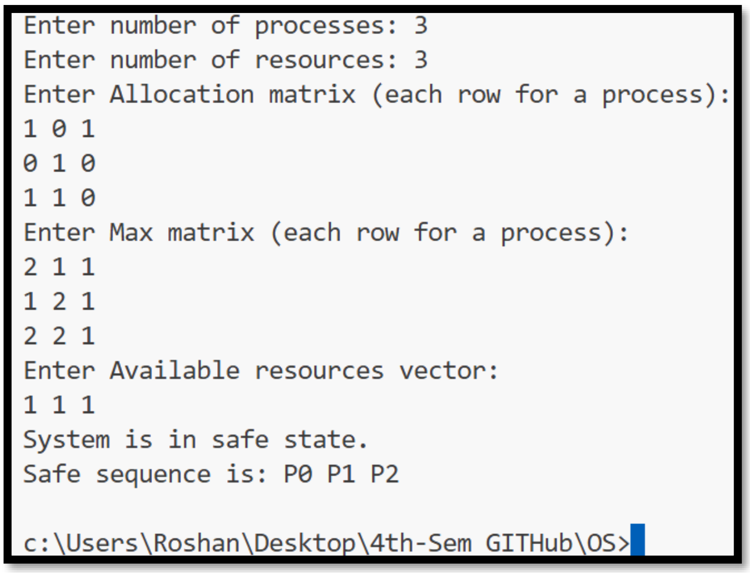
Output:

- Unsafe sequence output:



```
Enter number of processes: 3
Enter number of resources: 3
Enter Allocation matrix (each row for a process):
1 0 1
0 1 0
1 1 1
Enter Max matrix (each row for a process):
2 1 2
1 2 1
2 2 2
Enter Available resources vector:
1 0 0
System is NOT in safe state (Deadlock possible)

c:\Users\Roshan\Desktop\4th-Sem GITHub\OS>
```

- Safe sequence output:



```
Enter number of processes: 3
Enter number of resources: 3
Enter Allocation matrix (each row for a process):
1 0 1
0 1 0
1 1 0
Enter Max matrix (each row for a process):
2 1 1
1 2 1
2 2 1
Enter Available resources vector:
1 1 1
System is in safe state.
Safe sequence is: P0 P1 P2

c:\Users\Roshan\Desktop\4th-Sem GITHub\OS>
```

Conclusion:

The Deadlock Detection Algorithm helps in identifying whether a system is in a safe state or is experiencing deadlock. By analyzing resource allocation, maximum demands, and availability, the system can detect potential deadlocks before they cause process starvation or system halt. Using this approach ensures better resource management and system reliability.