Ganpat University
॥ विद्यया समाजोत्कर्षः ॥
Institute of Computer Technology

Name: Tushar Panchal

En.No: 21162101014

Sub: OS(Operating Systems)

Branch: CBA

Batch:41

----------PRACTICAL 8----------

❖ *Experiment-8 :*

To Understand Deadlock in OS & implement Deadlock avoidance algorithm.

Q1.:

Create any two Resource Allocation Graphs which may lead to deadlock and system in unsafe state of your choice. Demonstrate the RAGs using C code. (Hint : Sample RAG as Instructed by Instructor)

✓ Source Code :

```
#include <iostream>
#define P 3
#define R 2
using namespace std;
int main()
{
    int available[R] = {0, 0};
    int alloc[P][R] =
        {
            {1, 0}, // P0
            {0, 1}, // P1
            {0, 1}  // P2
        };

    int req[P][R] =
        {
            {0, 1}, // P0
            {1, 0}, // P1
            {1, 0}  // P2
```

```
        };
    int p_flag[P] = {};
    int flag = 0;
    bool safe = true;
    for (int n = 0; n < P; n++)
    {
        for (int i = 0; i < P; i++)
        {
            if (p_flag[i] == 0)
            {
                int flag = 0;
                for (int j = 0; j < R; j++)
                {
                    if (req[i][j] <= available[j])
                        flag++;
                }
                if (flag == R)
                {
                    cout << "Process" << i << "executed" << endl;
                    cout << "Available Resource: ";
                    for (int k = 0; k < R; k++)
                    {
                        available[k] += alloc[i][k];
                        cout << available[k] << " ";
                    }
                    cout << endl;
                    p_flag[i] = 1;
                    break;
                }
            }
        }
    }
    for (int i = 0; i < P; i++)
    {
        if (!p_flag[i])
        {
            safe = false;
            break;
        }
    }
    if (safe)
        cout << " System is in safe state" << endl;
    else
        cout << " System is in unsafe state" << endl;
    return 0;
}
```

- ✓ **Output :**

```
┌─tushar@tushar in ~/Documents/OS/8
└─λ ./1
System is in unsafe state
```

Q2.:

For a given Problem -1 , Design the solution after applying Banker Algorithm using C program.

✓ Source Code :

```
#include <iostream>
#define P 5
#define R 3
using namespace std;
int main()
{
    int available[R] = {2, 1, 0}; // Resources available initially
    int alloc[P][R] = {
        {1, 1, 2}, // P0
        {2, 1, 2}, // P1
        {4, 0, 1}, // P2
        {0, 2, 0}, // P3
        {1, 1, 2}  // P4
    };
    int max[P][R] = {
        {4, 3, 3}, // P0
        {3, 2, 2}, // P1
        {9, 0, 2}, // P2
        {7, 5, 3}, // P3
        {1, 1, 2}  // P4
    };
    int req[P][R];
    cout << " Resources Needed:" << endl;
    for (int i = 0; i < P; i++)
    {
        for (int j = 0; j < R; j++)
        {
            req[i][j] = max[i][j] - alloc[i][j];
            cout << req[i][j] << " ";
        }
        cout << endl;
    }
    int p_flag[P] = {};
    int flag = 0;
    for (int n = 0; n < P; n++)
    {
        for (int i = 0; i < P; i++)
        {
            if (p_flag[i] == 0)
            {
                int flag = 0;
                for (int j = 0; j < R; j++)
                {
                    if (req[i][j] <= available[j])
                    {
```

```
                    flag++;
                }
                if (flag == R)
                {
                    cout << " Process " << i << " Executed" << endl;
                    cout << "Available Resources : ";
                    for (int k = 0; k < R; k++)
                    {
                        available[k] += alloc[i][k];
                        cout << available[k] << " ";
                    }
                    cout << endl;
                    p_flag[i] = 1;
                    break;
                }
            }
        }
    }
}
```

✓ **Output :**

```
┌─tushar@tushar in ~/Documents/OS/8 took 1ms
└─λ ./2
Resources Needed:
3 2 1
1 1 0
5 0 1
7 3 3
0 0 0
Process 1 Executed
Available Resources
Process 4 Executed
Available Resources : 5 3 4
Process 0 Executed
Available Resources : 6 4 6
Process 2 Executed
Available Resources : 10 4 7
Process 3 Executed
Available Resources : 10 6 7
```