Fr. Conceicao Rodrigues College of Engineering Department of Computer Engineering			
Student's Roll No	**9913**	**Students Name**	**Mark Lopes**
Date of Performance		**SE Computer – Div**	**A**

Aim: To study Deadlock detection and Avoidance strategies

Lab Outcome:

CSL403.3: Understand and apply the concepts of synchronization and deadlocks

Pre-requirement: Python Programming.

Problem Statements:
WAP for the following.

Inputs: Number of processes, No of Resources, Instances of each resources, Number of resources held by each process , Number of resources needed by each process/Maximum number of resources needed by each process.

Write a menu driven program.

1) Detect if a deadlock exists. Also show the processes involved in deadlock

2) Check if the deadlock can be avoided (using bankers algo.). If yes, give the safe state sequence.

References:

https://www.javatpoint.com/os-resource-allocation-graph
https://www.javatpoint.com/os-deadlock-avoidance

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

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

// Function to calculate the need of each process
void calculateNeed(int need[P][R], int maxm[P][R], int allot[P][R])
{
    for (int i = 0; i < P; i++)
    {
        for (int j = 0; j < R; j++)
        {
            // Need of instance = maxm instance - allocated instance
            need[i][j] = maxm[i][j] - allot[i][j];
```

```
        }
    }
}

// Function to find if the system is in a safe state or not
bool isSafe(int processes[P], int avail[R], int maxm[P][R], int allot[P][R])
{
    int need[P][R];
    calculateNeed(need, maxm, allot);

    int finish[P] = {0};
    int safeSeq[P];
    int work[R];

    // Initialize work with available resources
    for (int i = 0; i < R; i++)
    {
        work[i] = avail[i];
    }

    int count = 0;
    while (count < P)
    {
        bool found = false;
        for (int p = 0; p < P; p++)
        {
            if (finish[p] == 0)
            {
                int j;
                for (j = 0; j < R; j++)
                {
                    if (need[p][j] > work[j])
                    {
                        break;
                    }
                }
                if (j == R)
                {
                    // Process p can be executed
                    for (int k = 0; k < R; k++)
                    {
                        work[k] += allot[p][k];
                    }
```

```
                    safeSeq[count++] = p;
                    finish[p] = 1;
                    found = true;
                }
            }
        }
        if (!found)
        {
            printf("System is not in a safe state.\n");
            return false;
        }
    }

    // If system is in a safe state, print safe sequence
    printf("System is in a safe state.\nSafe sequence is: ");
    for (int i = 0; i < P; i++)
    {
        printf("%d ", safeSeq[i]);
    }
    printf("\n");

    return true;
}

int main()
{
    int processes[] = {0, 1, 2, 3, 4};
    int avail[] = {3, 3, 2};

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

    int allot[P][R] = {
        {0, 1, 0},
        {2, 0, 0},
        {3, 0, 2},
        {2, 1, 1},
        {0, 0, 2}};
```

```
    // Check if system is in a safe state
    isSafe(processes, avail, maxm, allot);

    return 0;
}
```

```
System is in a safe state.
Safe sequence is: 1 3 4 0 2
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\Os>
```

On time Submission(2)	Knowledge of Topic(4)	Implementation and Demonstraion(4)	Total (10)
Signature of Faculty		**Date of Submission**	