**OS - Practical 8**

**Name: Rishabh Jain**

**Branch & Sem: CSE - IV**

**Sec: A**

**Roll no.:54**

**Aim:** Write C programs to demonstrate the Banker’s Algorithm and recovery processes.

**CODE:**

#include <stdio.h>

#include <stdbool.h>

#define MAX 5

bool finish[MAX] = {false};

int available[4] = {3, 3, 2, 1};

int ALLOCATION[MAX][4];

int MAXIMUM[MAX][4];

int NEED[MAX][4];

void input()

{

    printf("Enter ALLOCATION matrix\n");

    for (int i = 0; i < MAX; i++)

    {

        for (int j = 0; j < 4; j++)

        {

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

        }

    }

    printf("Enter MAXIMUM matrix\n");

    for (int i = 0; i < MAX; i++)

    {

        for (int j = 0; j < 4; j++)

        {

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

        }

    }

}

void calculateNeed()

{

    for (int i = 0; i < MAX; i++)

    {

        for (int j = 0; j < 4; j++)

        {

            NEED[i][j] = MAXIMUM[i][j] - ALLOCATION[i][j];

        }

    }

}

void output()

{

    printf("ALLOCATION Matrix is\n");

    for (int i = 0; i < MAX; i++)

    {

        for (int j = 0; j < 4; j++)

        {

            printf("%d\t", ALLOCATION[i][j]);

        }

        printf("\n");

    }

    printf("MAX Matrix is\n");

    for (int i = 0; i < MAX; i++)

    {

        for (int j = 0; j < 4; j++)

        {

            printf("%d\t", MAXIMUM[i][j]);

        }

        printf("\n");

    }

    printf("NEED Matrix is\n");

    for (int i = 0; i < MAX; i++)

    {

        for (int j = 0; j < 4; j++)

        {

            printf("%d\t", NEED[i][j]);

        }

        printf("\n");

    }

}

bool isSafe()

{

    int work[4];

    bool finishCopy[MAX];

    for (int i = 0; i < 4; i++)

    {

        work[i] = available[i];

    }

    for (int i = 0; i < MAX; i++)

    {

        finishCopy[i] = finish[i];

    }

    int count = 0;

    bool found = false;

    int safeSequence[MAX];

    int safeCount = 0;

    while (count < MAX)

    {

        found = false;

        for (int i = 0; i < MAX; i++)

        {

            if (!finishCopy[i])

            {

                bool canAllocate = true;

                for (int j = 0; j < 4; j++)

                {

                    if (NEED[i][j] > work[j])

                    {

                        canAllocate = false;

                        break;

                    }

                }

                if (canAllocate)

                {

                    for (int j = 0; j < 4; j++)

                    {

                        work[j] += ALLOCATION[i][j];

                    }

                    finishCopy[i] = true;

                    found = true;

                    count++;

                    safeSequence[safeCount++] = i;

                }

            }

        }

        if (!found)

        {

            return false;

        }

    }

    printf("Available after iteration %d:\n", count);

    for (int i = 0; i < 4; i++)

    {

        printf("%d\t", work[i]);

    }

    printf("\n");

    printf("Safe sequence: ");

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

    {

        printf("P%d ", safeSequence[i]);

    }

    printf("\n");

    return true;

}

void recoveryAlgorithm()

{

    int process;

    printf("Enter the process number to recover: ");

    scanf("%d", &process);

    if (process < 0 || process >= MAX)

    {

        printf("Invalid process number.\n");

        return;

    }

    if (finish[process] == true)

    {

        printf("Process P%d is already finished. No recovery needed.\n", process);

        return;

    }

    printf("Recovering process P%d...\n", process);

    for (int j = 0; j < 4; j++)

    {

        available[j] += ALLOCATION[process][j];

        ALLOCATION[process][j] = 0;

        NEED[process][j] = 0;

    }

    finish[process] = true;

    printf("Process P%d has been recovered.\n", process);

}

int main()

{

    input();

    calculateNeed();

    output();

    if (isSafe())

    {

        printf("The system is in a safe state.\n");

    }

    else

    {

        printf("The system is not in a safe state.\n");

        recoveryAlgorithm();

        if (isSafe())

        {

            printf("The system has been successfully recovered and is now in a safe state.\n");

        }

        else

        {

            printf("The system could not be recovered and is still not in a safe state.\n");

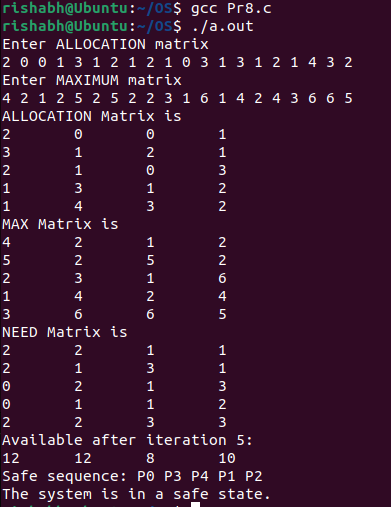
        }

    }

    return 0;

}

**OUTPUT:**



**RESULT:** We LearnedC program to demonstrate the Banker’s Algorithm and recovery Processes has been implemented.