Operating System

Banker's Algorithm

Program: communication and electronics.

Section: 8

Name	ID
Mohamed Nasser Mohamed Ibrahim	18011632
Mohamed Badr Saad Zaghlool Mohamed	18015047
Mohammed Adel Omar Bayoumy	18011514
Seif Mohamed Ashraf	18015038
Ziad Mahmmoud Fawzy	18010745

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#define true 1
#define false 0
#define processNum 5
#define resourceNum 3
typedef int bool;
int available[resourceNum];
int maxRequest[processNum][resourceNum];
int allocation[processNum][resourceNum];
int need[processNum][resourceNum];
bool Finish[processNum];
int safeSeries[processNum];
int request[resourceNum];
 int *readtxt(int a)
{
    FILE *myFile;
    switch (a)
    {
    case 1:
    myFile = fopen("TestFile1.txt", "r");  //to read a text file
        break;
    default:
    myFile = fopen("TestFile2.txt", "r");
        break;
    }
    static int numberArray[33];
    int i;
    for (i = 0; i < 33; i++)
    {
        fscanf(myFile, "%1d", &numberArray[i]);
    }
    return numberArray;
}

void Init(int x)
{
    int i, j;
    int k=0;
    int *numberArray=readtxt(x);
    for (int i=0;i<5;i++)
```

```
    {
    for (int j=0;j<3;j++)
    {
        allocation[i][j]=numberArray[k++];
    }
    }
    for (int i=0;i<5;i++)
    {
    for (int j=0;j<3;j++)
    {
        maxRequest[i][j]=numberArray[k++];
    }
    }
    for (int j=0;j<3;j++)
    {
        available[j]=numberArray[k++];
    }
    for (i = 0; i < processNum; i++) {
        for (j = 0; j < resourceNum; j++) {
            need[i][j]=maxRequest[i][j]-allocation[i][j];
        }
    }
}

void showInfo()
{
    int i, j;
    printf("Current resource remaining: ");
    for (j = 0; j < resourceNum; j++) {
        printf("%d ", available[j]);
    }
    printf("\n");
    printf(" PID\t Max\t\tAllocation\tNeed\n");
    for (i = 0; i < processNum; i++) {
        printf(" P%d\t", i);
        for (j = 0; j < resourceNum; j++) {
            printf("%d ", maxRequest[i][j]);
        }
        printf("\t\t");
        for (j = 0; j < resourceNum; j++) {
            printf("%d ", allocation[i][j]);
        }
        printf("\t\t");
        for (j = 0; j < resourceNum; j++) {
            printf("%d ", need[i][j]);
```

```
        }
        printf("\n");
    }
}

void SafeInfo(int *work, int i)
{
    int j;
    printf(" P%d\t", i);
    for (j = 0; j < resourceNum; j++) {
        printf("%d ", work[j]);
    }
    printf("\t\t");
    for (j = 0; j < resourceNum; j++) {
        printf("%d ", need[i][j]);
    }
    printf("\t ");
    for (j = 0; j < resourceNum; j++) {
        printf("%d ", allocation[i][j]);
    }
    printf("\t\t");
    for (j = 0; j < resourceNum; j++) {
        printf("%d ", allocation[i][j] + work[j]);
    }
    printf("\n");
}
bool isSafe()
{
    int i, j, k, flag, temp0;
    int trueFinished = 0;
    int work[resourceNum];
    for (i = 0; i < resourceNum; i++) {
        work[i] = available[i];
    }

    for (i = 0; i < processNum; i++) {
        Finish[i] = false;
    }
    i = 0;
    int temp = 0;
    while (trueFinished != processNum) {
        int j = 0;
        if (Finish[i] != true) {
            for (j = 0; j < resourceNum; j++) {
                if (need[i][j] > work[j]) { break; }
```

```
            }
        }
        if (j == resourceNum) {
            Finish[i] = true;
            SafeInfo(work, i);
            for (k = 0; k < resourceNum; k++) {
                work[k] += allocation[i][k];
            }
            int k2;
            safeSeries[trueFinished++] = i;
        }
        i++;
        if (i >= processNum)
        {
            if (flag == 0)
            {
                temp = trueFinished;
                temp0 = trueFinished;
            }
            i = i % processNum;
            if (flag == 1) {
                temp = trueFinished;
                if (temp == temp0)
                    break;
                else
                    temp0 = temp;
            }
            flag = 1;
        }
        temp = trueFinished;
    }

    if (trueFinished == processNum) {
        printf("\nsystem safe! \n\nThe safe sequence is! ");
        for (i = 0; i < processNum; i++) {
            printf("%d ", safeSeries[i]);
        }
        return true;
    }
    printf("******system unsafe! ******\n");
    return false;
}


int main()
```

```
{
    int i;
    while (true){
    printf("\nEnter the Test case number (enter 999 if you want to exit): ");
    scanf("%d",&i);
    if(i==999){
        break;
    }
    if((i!=1)&&(i!=2))
    {
        printf("\nIncorrect Input !!!\n");
        continue;
    }
    Init(i);
    printf("------------------------------------------------------------\n");
    showInfo();
    printf("\nSystem safety analysis\n");
    printf(" PID\t Work\t\tNeed\tAllocation\tWork+Allocation\n");
    isSafe();
    }
    return 0;
}
```

Output in test case 1

```
Enter the Test case number (enter 999 if you want to exit): 1
-----------------------------------------------------------
Current resource remaining: 3 3 2
 PID     Max            Allocation      Need
 P0     7 5 3           0 1 0           7 4 3
 P1     3 2 2           2 0 0           1 2 2
 P2     9 0 2           3 0 2           6 0 0
 P3     2 2 2           2 1 1           0 1 1
 P4     4 3 3           0 0 2           4 3 1

System safety analysis
 PID     Work           Need    Allocation      Work+Allocation
 P1     3 3 2           1 2 2    2 0 0          5 3 2
 P3     5 3 2           0 1 1    2 1 1          7 4 3
 P4     7 4 3           4 3 1    0 0 2          7 4 5
 P0     7 4 5           7 4 3    0 1 0          7 5 5
 P2     7 5 5           6 0 0    3 0 2          10 5 7

system safe!

The safe sequence is! 1 3 4 0 2
```

Output in test case 2

```
Enter the Test case number (enter 999 if you want to exit): 2
-----------------------------------------------------------
Current resource remaining: 2 3 0
 PID     Max            Allocation      Need
 P0     7 5 3           0 1 0           7 4 3
 P1     3 2 2           3 0 2           0 2 0
 P2     9 0 1           3 0 1           6 0 0
 P3     2 2 2           2 1 1           0 1 1
 P4     4 3 3           0 0 2           4 3 1

System safety analysis
 PID     Work           Need    Allocation      Work+Allocation
 P1     2 3 0           0 2 0    3 0 2          5 3 2
 P3     5 3 2           0 1 1    2 1 1          7 4 3
 P4     7 4 3           4 3 1    0 0 2          7 4 5
 P0     7 4 5           7 4 3    0 1 0          7 5 5
 P2     7 5 5           6 0 0    3 0 1          10 5 6

system safe!

The safe sequence is! 1 3 4 0 2
```