参考代码二　银行家算法

1.说明

本程序为避免进程死锁的银行家算法。其功能包括：

(1)检查 T_0 时刻的安全性；

(2)判断进程所申请的资源能否分配。

2.数据结构

本程序用到了下列数据结构：

(1)全局变量

```
int Available[m]={3,3,2};
int Max[n][m]={{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};
int Allocation[n][m]={{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}};
int Need[n][m]={{7,4,3},{1,2,2},{6,0,0},{0,1,1},{4,3,1}};
int Work[m];
int Finish[n]={0,0,0,0,0};
int Order[n]={0,0,0,0,0};
int Work_Rec[n+1][m];
int Work_Allocation[n][m];
```

(2)宏定义

```
#define m 3
#define n 5
```

3.程序及运行结果

(1)程序清单　(C 语言编写，在 VC 6.0 环境下运行)

```
//********************************************************************************
//*******************            参考代码二 银行家算法            ********************
//*******************            主程序文件 Banker.c             ********************
//*******************            版权所有: 陈礼青                ********************
//********************************************************************************

#include <stdio.h>

#define m 3
#define n 5

int Available[m]={3,3,2};
int Max[n][m]={{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};
int Allocation[n][m]={{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}};
int Need[n][m]={{7,4,3},{1,2,2},{6,0,0},{0,1,1},{4,3,1}};
int Work[m];
int Finish[n]={0,0,0,0,0};
int Order[n]={0,0,0,0,0};
int Work_Rec[n+1][m];
int Work_Allocation[n][m];
```

```
void Init_Finish()
{
    int i=0;
    for(;i<n;i++)
        Finish[i]=0;
}

/*显示资源分配表*/
void Dis_Res_Table()
{
    int i,j;
    printf("\n*******目前资源分配表********\n");
    printf("资源情况 Max          Allocation      Need          Available\n");
    printf("进程 A    B    C    A    B    C    A    B    C    A    B    C\n");
    for(i=0;i<n;i++)
    {
        printf("P%-4d",i);
        for(j=0;j<m;j++)
            printf("%-4d",Max[i][j]);
        for(j=0;j<m;j++)
            printf("%-4d",Allocation[i][j]);
        for(j=0;j<m;j++)
            printf("%-4d",Need[i][j]);
        for(j=0;i==0&&j<3;j++)
            printf("%-4d",Available[j]);
        printf("\n");
    }
}

/*显示安全序列分析表*/
void Dis_Safe_Sequence_Table()
{
    int i,j,k;
    printf("**********安全序列分析表**********\n");
    printf("资源情况 Work        Need          Allocation      Work+Allocation    Finish\n");
    printf("进程 A    B    C    A    B    C    A    B    C         A        B        C\n");
    for(i=0;i<n;i++)
    {
        k=Order[i];
        printf("P%-4d",k);
        for(j=0;j<m;j++)
            if(i==0)
                printf("%-4d",Available[j]);
            else
                printf("%-4d",Work_Rec[i][j]);
        for(j=0;j<m;j++)
```

```
            printf("%-4d",Need[k][j]);
        for(j=0;j<m;j++)
            printf("%-4d",Allocation[k][j]);
        printf("      ");
        for(j=0;j<m;j++)
            printf("%-6d",Work_Allocation[k][j]);
        printf("%-4d",Finish[j]);
        printf("\n");
    }
}

void Try_Allocation(int i,int *Request)
{
    int j;
    for(j=0;j<m;j++)
    {
        Available[j]=Available[j]-Request[j];
        Allocation[i][j]=Allocation[i][j]+Request[j];
        Need[i][j]=Need[i][j]-Request[j];
    }
}

void Undo(int i,int *Request)
{
    int j;
    for(j=0;j<m;j++)
    {
        Available[j]=Available[j]+Request[j];
        Allocation[i][j]=Allocation[i][j]-Request[j];
        Need[i][j]=Need[i][j]+Request[j];
    }
}

void Dis_Safe_Order()
{
    int i;
    printf("安全序列：");
    for(i=0;i<n;i++)
        printf("P%d ",Order[i]);
    printf("\n");
}

int Small(int *pre,int *last)
{
    int i;
    for(i=0;i<m;i++)
```

```
            if(pre[i]>last[i]) return 0;
        return 1;
}

void Add(int *pre,int *last)
{
        int i;
        for(i=0;i<m;i++)
            pre[i]+=last[i];
}

void Assign(int *pre,int *last)
{
        int i;
        for(i=0;i<m;i++)
            pre[i]=last[i];
}

/*安全性算法*/
int Safer()
{
        int i,modified=0,k=0;
        Init_Finish();
        Assign(Work,Available);
        while(Finish[0]!=1||Finish[1]!=1||Finish[2]!=1||Finish[3]!=1||Finish[4]!=1)
        {
            for(i=0;i<n;i++)
                if(Finish[i]==0&&Small(Need[i],Work))
                {
                    Add(Work,Allocation[i]);
                    Finish[i]=1;
                    Order[k++]=i;
                    Assign(Work_Allocation[i],Work);
                    Assign(Work_Rec[k],Work);
                    modified=1;
                    break;
                }
            if(modified==0) return 0;
            modified=0;
        }
        if((Finish[0]==1&&Finish[1]==1&&Finish[2]==1&&Finish[3]==1&&Finish[4]==1))
            Dis_Safe_Order();
        return 1;
}
```

```
/*银行家算法*/
void Banker(int i,int *Request)
{
    if(Small(Request,Need[i])&&Small(Request,Available))
    {
        Try_Allocation(i,Request);
        if(!Safer())
        {
            Undo(i,Request);
            printf("\n 无安全序列，让 P%d 等待 .....\n\n",i);
        }
        else
            Dis_Safe_Sequence_Table();
    }
    else
        printf("\n 系统中无足够资源，让 P%d 等待 ...\n\n",i);
}

void main()
{
    int Request1[m]={1,0,2};
    int Request4[m]={3,3,0};
    int Request0[m]={0,2,0};

/*1.T0 时刻的资源分配表*/
    printf("\n*******1.T0 时刻的资源分配表********\n");
    Dis_Res_Table();
    getchar();

/*2.T0 时刻的安全性*/
    printf("\n*******2.T0 时刻的安全性********\n");
    if(Safer()) Dis_Safe_Sequence_Table();
    getchar();

/*3.P1 请求资源*/
    printf("\n*******3.P1 请求资源 Request(1,0,2)******\n");
    Banker(1,Request1);
    getchar();

/*4.P4 请求资源*/
    printf("\n*******4.P4 请求资源 Request(3,3,0)******\n");
    Banker(4,Request4);
    getchar();
    Dis_Res_Table();
    getchar();
```

```
/*5.P0 请求资源*/
    printf("\n*******5.P0 请求资源 Request(0,2,0)******\n");
    Banker(0,Request0);
    getchar();

/*6.P0 请求资源*/
    printf("\n*******6.P0 请求资源 Request(0,1,0)******\n");
    Request0[1]=1;
    Banker(0,Request0);
    Dis_Res_Table();
    getchar();
}
```

(2)运行结果

在本程序运行过程中，首先打印 T0 时刻的资源分析表，按下任何键后，得到 T0 时刻的安全性分析表，再按下任何键，可分别打印 P1,P4,P0 请求资源的安全性检查过程。具体的运行结果略。