实验 3　银行家算法

一、实验目的

通过本实验加深对死锁、安全状态、安全序列等概念的理解，熟悉死锁避免方法的经典算法银行家算法的设计。

二、实验内容

1. 模拟实现利用银行家算法进行资源分配。

三、实验步骤

1. 银行家算法 1

该算法的重点在于模拟实现银行家算法，功能较为简单。其主要代码如下所示。

```
#include <iostream.h>

#define n 5                    /*m 个资源,n 个进程*/
#define m 3
int Available[m] = {3,3,2};    /*可用资源数组*/
int Max[n][m] = {{7,5,3},{3,2,2},{9,0,2},{2,2,2},{4,3,3}};      /*
最大需求矩阵*/
int Allocation[n][m] = {{0,1,0},{2,0,0},{3,0,2},{2,1,1},{0,0,2}};
    /*分配矩阵*/
int Need[n][m] = {{7,4,3},{1,2,2},{6,0,0},{0,1,1},{4,3,1}};     /*
需求矩阵*/
int Request[n][m];             /*进程需要资源数*/
bool Finish[n];                /*系统是否有足够的资源分配*/
int P[n];                      /*安全序列*/

bool Safe();
void Bank();

void main()
{
    Safe();
    Bank();
}

void Bank()                    /*银行家算法*/
{
    int i,j,pno;
```

```
    char again;
    bool input_again = false;
    while(1)
    {
        cout<<"请输入要申请资源的进程号(注:第 1 个进程号为 0,依次类
推)"<<endl;
        cin>>pno;

        cout<<"请输入进程所请求的各类资源的数量"<<endl;
        for(j=0;j<m;j++)
        {
            cin>>Request[pno][j];
        }

        for(j=0;j<m;j++)
        {
            // 判断前两个条件
            if(Request[pno][j]>Need[pno][j])
            {
                cout<<"您输入的请求数超过进程的需求量!请重新输入!"<<endl;
                input_again = true;
                break;
            }
            if(Request[pno][j]>Available[j])
            {
                cout<<"您输入的请求数超过系统有的资源数!请重新输入!"<<endl;
                input_again = true;
                break;
            }
        }
        if(input_again)
        {
            continue;
        }
        // 试探分配资源
        for(j=0;j<m;j++)
        {
            Available[j] -= Request[pno][j];
            Allocation[pno][j] += Request[pno][j];
            Need[pno][j] -= Request[pno][j];
        }
        // 检查系统安全性
        if(Safe())
        {
```

```
            cout<<"同意分配请求!"<<endl;
        }
        else
        {
            cout<<"您的请求被拒绝!"<<endl;
            for(j=0;j<m;j++)
            {
                Available[j] += Request[pno][j];
                Allocation[pno][j] -= Request[pno][j];
                Need[pno][j] += Request[pno][j];
            }
        }

        for(i=0;i<n;i++)
        {
            Finish[i]=false;
        }

        cout<<"您还想再次请求分配吗?是请按 y/Y,否请按其它键"<<endl;
        cin>>again;
        if(again=='y'||again=='Y')
        {
            continue;
        }
        break;
        }
}

bool Safe()          /*安全性算法*/
{
    // 初始化,设置两个向量
    int i,j,k,l=0;
    int Work[m];
    for(j=0;j<m;j++)
    {
        Work[j]=Available[j];
    }
    for(i=0;i<n;i++)
    {
        Finish[i]=false;
    }

    for(i=0;i<n;i++)
    {
```

```
        if(Finish[i]==true)
        {
            continue;
        }
        else
        {
            for(j=0;j<m;j++)
            {
                if(Need[i][j]>Work[j])
                {
                    break;
                }
            }
            if(j==m)
            {
                Finish[i]=true;
                for(k=0;k<m;k++)
                {
                    Work[k]+=Allocation[i][k];
                }
                P[l++]=i;
                i=-1;          // 重新开始查找符合条件的进程
            }
            else
            {
                continue;      // 查询下一个进程
            }
        }
        if(l==n)
        {
            cout<<"系统是安全的"<<endl;
            cout<<"安全序列:"<<endl;
            for(i=0;i<l;i++)
            {
                cout<<P[i];
                if(i!=l-1)
                {
                    cout<<"-->";
                }
            }
            cout<<""<<endl;
            return true;
        }
    }
```

```
    cout<<"系统是不安全的!"<<endl;
    return false;
}
```

2. 银行家算法 2

该算法对银行家算法进行了补充，功能有所完善。其主要代码如下所示。

```
#include <iostream.h>

#define MAXPROCESS 10                          /*最大进程数*/
#define MAXRESOURCE 100                        /*最大资源数*/
int Available[MAXRESOURCE];                    /*可用资源数组*/
int Max[MAXPROCESS][MAXRESOURCE];              /*最大需求矩阵*/
int Allocation[MAXPROCESS][MAXRESOURCE];       /*分配矩阵*/
int Need[MAXPROCESS][MAXRESOURCE];             /*需求矩阵*/
int Request[MAXPROCESS][MAXRESOURCE];          /*进程需要资源数*/
bool Finish[MAXPROCESS];                     /*系统是否有足够的资源分配*/
int P[MAXPROCESS];                           /*记录序列*/
int m;                                       /*m 个资源,n 个进程*/
int n;

void Init();
bool Safe();
void Bank();
void main()
{
    Init();
    Safe();
    Bank();
}

void Init()                        //初始化算法
{
    int i,j;
    cout<<"请输入进程的数目: ";
    cin>>n;
    cout<<"请输入资源的种类: ";
    cin>>m;

    cout<<"请输入每个进程最多所需的各资源数,按照"<<n<<" x "<<m<<"矩阵输入
"<<endl;
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
```

```
            cin>>Max[i][j];
        }
    }

    cout<<"请输入每个进程已分配的各资源数,也按照"<<n<<" x "<<m<<"矩阵输入
"<<endl;
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
            cin>>Allocation[i][j];
            Need[i][j]=Max[i][j]-Allocation[i][j];
            if(Need[i][j]<0)
            {
                cout<<"您输入的第"<<i+1<<"个进程所拥有的第"<<j+1<<"类资源
数错误,请重新输入:"<<endl;
                j--;
                continue;
            }
        }
    }

    cout<<"请输入各个资源现有的数目:"<<endl;
    for(j=0;j<m;j++)
    {
        cin>>Available[j];
    }
}
```

```
        }

        for(j=0;j<m;j++)
        {
            // 判断前两个条件
```

```
        //判断输入条件
        if(Request[pno][j]>Need[pno][j])
        {
            cout<<"您输入的请求数超过进程的需求量!请重新输入!"<<endl;
            input_again = true;
            break;
        }
        if(Request[pno][j]>Available[j])
        {
            cout<<"您输入的请求数超过系统有的资源数!请重新输入!"<<endl;
            input_again = true;
            break;
        }
    }
    if(input_again)
    {
        continue;
    }
    // 试探分配资源
    for(j=0;j<m;j++)
    {
        Available[j] -= Request[pno][j];
        Allocation[pno][j] += Request[pno][j];
        Need[pno][j] -= Request[pno][j];
    }
    // 检查系统安全性
    if(Safe())
    {
        cout<<"同意分配请求!"<<endl;
    }
    else
    {
        cout<<"您的请求被拒绝!"<<endl;
        for(j=0;j<m;j++)
        {
            Available[j] += Request[pno][j];
            Allocation[pno][j] -= Request[pno][j];
            Need[pno][j] += Request[pno][j];
        }
    }
```

```
    for(i=0;i<n;i++)
    {
        Finish[i]=false;
```

```
            Finish[i]=false;
        }

        cout<<"您还想再次请求分配吗?是请按 y/Y,否请按其它键"<<endl;
        cin>>again;
        if(again=='y'||again=='Y')
        {
            continue;
        }
        break;
        }
}

bool Safe()                 /*安全性算法*/
{
    // 初始化,设置两个向量
    int i,j,k,l=0;
    int Work[MAXRESOURCE];
    for(j=0;j<m;j++)
    {
        Work[j]=Available[j];
    }
    for(i=0;i<n;i++)
    {
        Finish[i]=false;
    }

    for(i=0;i<n;i++)
    {
        if(Finish[i]==true)
        {
            continue;
        }
        else
        {
            for(j=0;j<m;j++)
            {
                if(Need[i][j]>Work[j])
                {
                    break;
                }
```

```
            }
            if(j==m)
```

```
            {
                Finish[i]=true;
                for(k=0;k<m;k++)
                {
                    Work[k]+=Allocation[i][k];
                }
                P[l++]=i;
                i=-1;      // 重新开始查找符合条件的进程
            }
            else
            {
                continue; // 查询下一个进程
            }
        }
        if(l==n)
        {
            cout<<"系统是安全的"<<endl;
            cout<<"安全序列:"<<endl;
            for(i=0;i<l;i++)
            {
                cout<<P[i];
                if(i!=l-1)
                {
                    cout<<"-->";
                }
            }
            cout<<""<<endl;
            return true;
        }
    }
    cout<<"系统是不安全的"<<endl;
    return false;
}
```

四、作业

1. 模拟实现利用银行家算法进行资源分配。