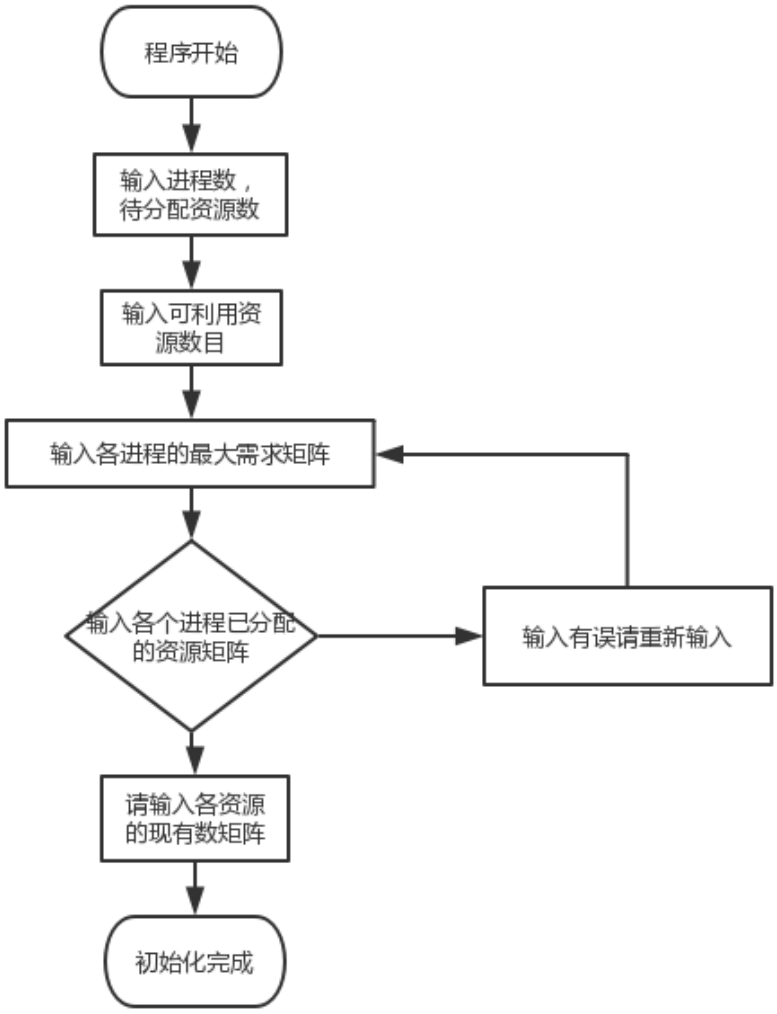
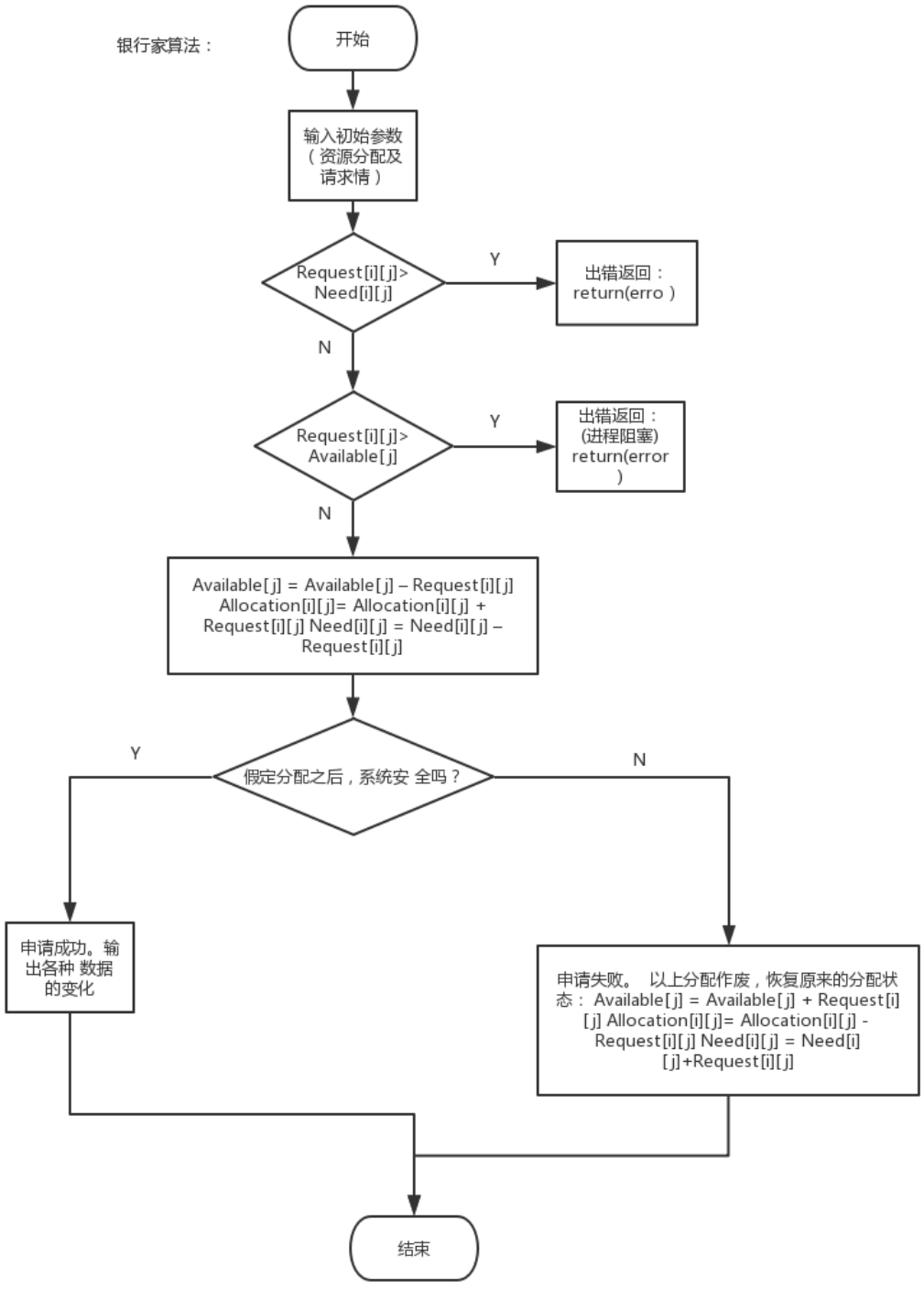
一、程序流程和源程序

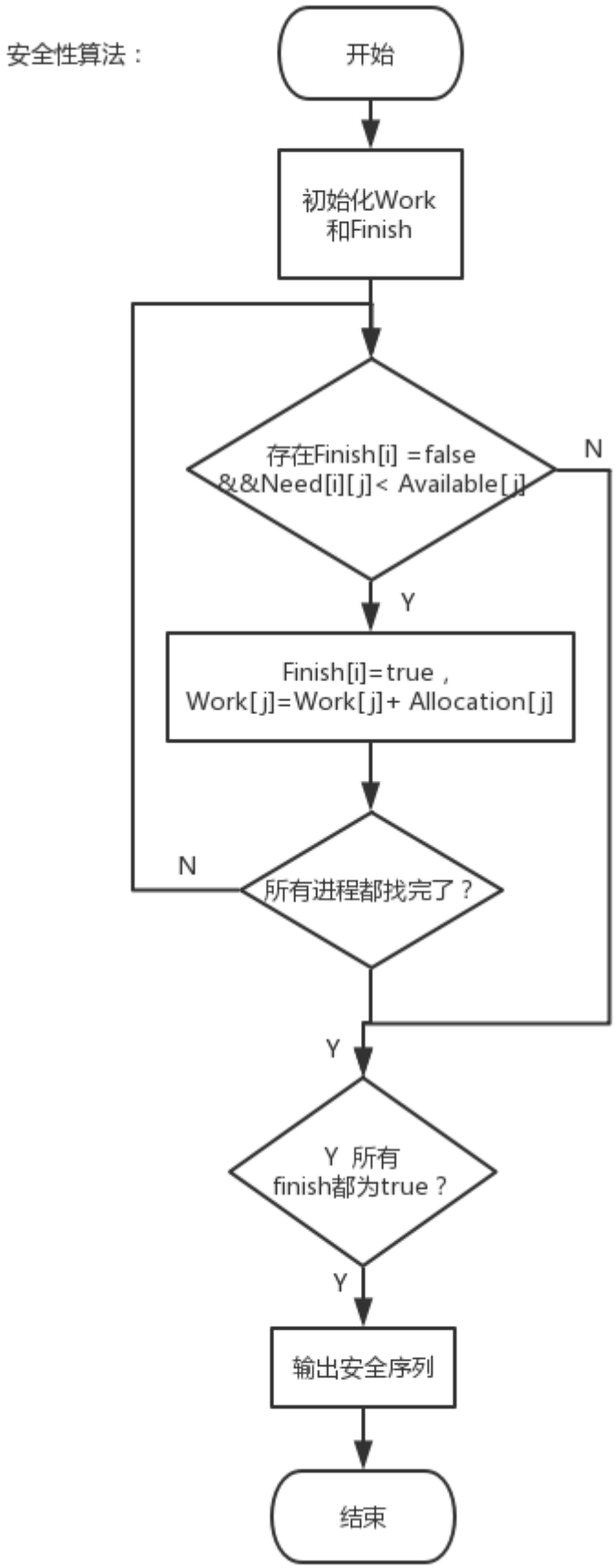
初始化：



银行家算法：


开始
输入初始参数（资源分配及请求情）
Request[i][j]> Need[i][j]
Y
出错返回：return(erro)
N
Request[i][j]> Available[j]
Y
出错返回：(进程阻塞) return(error)
N
Available[j] = Available[j] – Request[i][j] Allocation[i][j]= Allocation[i][j] + Request[i][j] Need[i][j] = Need[i][j] – Request[i][j]
假定分配之后，系统安 全吗？
Y
N
申请成功。输出各种 数据的变化
申请失败。 以上分配作废，恢复原来的分配状态： Available[j] = Available[j] + Request[i][j] Allocation[i][j]= Allocation[i][j] - Request[i][j] Need[i][j] = Need[i][j]+Request[i][j]
结束

安全性算法：


开始
初始化Work
和Finish
存在Finish[i] =false
&&Need[i][j]< Available[j]
N
Y
Finish[i]=true ,
Work[j]=Work[j]+ Allocation[j]
所有进程都找完了？
N
Y
Y 所有
finish都为true？
Y
输出安全序列
结束

```
/*************************
姓名：田松岩
班级：2015级4班
学号：2015015335
任务：银行家算法的模拟实现
*************************/
#include <iostream>
#include <string>
using namespace std;

#define MAXPROCESS  50  //所能执行的进程最大数
#define MAXRESOURCE 50  //资源最大数

//银行家算法中的数据结构
int Available[MAXRESOURCE];             //可用资源向量
int Max[MAXPROCESS][MAXRESOURCE];       //最大需求矩阵
int Allocation[MAXPROCESS][MAXRESOURCE];//已分配矩阵
int Need[MAXPROCESS][MAXRESOURCE];      //需求矩阵
int Request[MAXPROCESS][MAXRESOURCE];   //请求向量

//安全性算法中的数据结构
int Work[MAXRESOURCE];      //工作向量
bool Finish[MAXPROCESS];    //表示是否有足够的资源分配给进程
int SafeSeries[MAXPROCESS]; //安全序列

int n;  //当前系统中的进程数
int m;  //当前系统中的资源数

//算法初始化
void InitAlgorithm()
{
    cout << "请输入系统所要运行的进程总数:";
    cin >> n;
    cout << "请输入系统中分配的资源种类数：";
    cin >> m;
    int i, j;
```

```
    //可利用资源向量的初始化
    cout << "请分别输入" << m << "类资源的当前可利用资源数目:";
    for (i = 0; i < m; ++i)
    {
        cin >> Available[i];
    }

    //最大需求矩阵的初始化
    cout << "请输入各进程对各资源的最大需求矩阵(按"
        << n << "*" << m << "矩阵格式输入):" << endl;
    for (i = 0; i < n; ++i)
    {
        for (j = 0; j < m; ++j)
        {
            cin >> Max[i][j];
        }
    }

    //分配矩阵的初始化
    cout << "请输入分配矩阵(按"
        << n << "*" << m << "矩阵格式输入):" << endl;
    for (i = 0; i < n; ++i)
    {
        for (j = 0; j < m; ++j)
        {
            cin >> Allocation[i][j];
        }
    }

    //需求矩阵的初始化
    cout << "请输入此刻的需求矩阵(按"
        << n << "*" << m << "矩阵格式输入):" << endl;
    for (i = 0; i < n; ++i)
    {
        for (j = 0; j < m; ++j)
        {
            cin >> Need[i][j];
        }
    }

}
```

```
//输出资源分配情况
void PrintSrcAlloc()
{
    int i, j;
    for (i = 0; i < n; ++i)
    {
        cout << "进程P" << i << "的总体分配情况:" << endl;
        cout << "\tMax: ";
        for (j = 0; j < m; ++j)
        {
            cout << Max[i][j] << " ";
        }
        cout << endl;

        cout << "\tAllocation: ";
        for (j = 0; j < m; ++j)
        {
            cout << Allocation[i][j] << " ";
        }
        cout << endl;

        cout << "\tNeed: ";
        for (j = 0; j < m; ++j)
        {
            cout << Need[i][j] << " ";
        }
        cout << endl;
        cout << endl;
    }

    cout << "可利用资源情况: ";
    for (i = 0; i < m; ++i)
    {
        cout << Available[i] << " ";
    }
    cout << endl;
    cout << endl;

}
```

```
//输出此时进程iProcess利用安全性算法的分析情况
void PrintSafeSeries(int iProcess)
{
    int j;
    cout << "进程P" << iProcess << "的安全性分析情况:" << endl;
    cout << "\tWork: ";
    for (j = 0; j < m; ++j)
    {
        cout << Work[j] << " ";
    }
    cout << endl;

    cout << "\tNeed: ";
    for (j = 0; j < m; ++j)
    {
        cout << Need[iProcess][j] << " ";
    }
    cout << endl;

    cout << "\tAllocation: ";
    for (j = 0; j < m; ++j)
    {
        cout << Allocation[iProcess][j] << " ";
    }
    cout << endl;

    cout << "\tWork + Allocation: ";
    for (j = 0; j < m; ++j)
    {
        cout << Work[j] + Allocation[iProcess][j] << " ";
    }
    cout << endl;

    cout << "Finish[" << iProcess << "] = " << Finish[iProcess] << endl;
    cout << endl;
}
```

```
//判断是否存在安全序列
bool IsSafe()
{
    int i;
    for (i = 0; i < n; ++i)
    {
        if (!Finish[i])
            return false;  //不存在安全序列
    }
    return true;//存在安全序列
}

//界面
void Menu()
{
        cout << "================================" << endl;
        cout << "|   姓名：田松岩                |" << endl;
        cout << "|   班级：2015级4班             |" << endl;
        cout << "|   学号：2015015335            |" << endl;
        cout << "|   任务：银行家算法的模拟实现  |" << endl;
        cout << "================================" << endl;
}


//选出满足条件的进程
//函数返回进程编号
//若不存在满足条件的，则返回false,否则返回true
bool SelectProcess(int &iProcNum)
{
    iProcNum = -1;
    int i, j;
    for (i = 0; i < n; ++i)
    {
        if (Finish[i])  //Finish[i] != false
        {
            continue;
        }
        for (j = 0; j < m; ++j)
        {
            if (Need[i][j] > Work[j])
            {
                break;
            }
        }
```

```
            if (j == m)  //满足条件
            {
                iProcNum = i;
                return true;
            }
        }
        return false;
}

//安全性算法
bool SafeAlgorithm()
{
    int i, j;

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

    //初始化工作向量
    for (j = 0; j < m; ++j)
    {
        Work[j] = Available[j];
    }

    int iProcNum;
    //选择满足条件的进程
    i = 0;
    while (SelectProcess(iProcNum))
    {
        Finish[iProcNum] = true;
        PrintSafeSeries(iProcNum);  //输出此时利用安全性算法的分析情况
        int k;
        for (k = 0; k < m; ++k)
        {
            Work[k] += Allocation[iProcNum][k];
        }
        SafeSeries[i++] = iProcNum;  //进程号加入安全序列数组
    }

    if (IsSafe())
    {
        return true;  //存在一个安全序列
```

```
    }

    return false;  //不存在一个安全序列
}


//银行家算法
void BankAlgorithm()
{
    int i, j;
    cout << "请输入请求分配的进程号(0 - " << n - 1 << "): ";
    cin >> i;
    cout << "请依次输入进程P" << i << "对" << m << "类资源的请求分配量: ";
    for (j = 0; j < m; ++j)
    {
        cin >> Request[i][j];
    }

    //步骤一
    for (j = 0; j < m; ++j)
    {
        if (Request[i][j] > Need[i][j])
        {
            cout << "请求的资源已超过该资源宣布的最大值，请求资源失败!!" << endl;
            return;
        }
    }

    //步骤二
    for (j = 0; j < m; ++j)
    {
        if (Request[i][j] > Available[j])
        {
            cout << "没有足够的资源，请求资源失败!!" << endl;
            return;
        }
    }
```

```
    //步骤三
    //系统试探着把资源分配给进程Pi，并修改相应的数值
    for (j = 0; j < m; ++j)
    {
        Available[j] -= Request[i][j];
        Allocation[i][j] += Request[i][j];
        Need[i][j] -= Request[i][j];
    }

    //步骤四
    //系统执行安全性算法
    if (!SafeAlgorithm()) //检测到不安全，则恢复原来的状态
    {
        for (j = 0; j < m; ++j)
        {
            Available[j] += Request[i][j];
            Allocation[i][j] -= Request[i][j];
            Need[i][j] += Request[i][j];
        }
    }

}
```

```
//主函数调用
int main()
{
    Menu();
    InitAlgorithm();
    PrintSrcAlloc();  //打印当前资源情况
    system("pause");
    SafeAlgorithm();
    while (1)
    {
        string flag;
        if (IsSafe())
        {
            //存在安全序列
            cout << "恭喜！！资源分配成功!!" << endl;
            int i;
            cout << "此时刻存在的一个安全序列为：";
            for (i = 0; i < n - 1; ++i)
            {
                cout << "P" << SafeSeries[i] << "-->";
            }
            cout << "P" << SafeSeries[i] << endl;
            cout << "继续操作吗？[Y/N]:";
            cin >> flag;
        }
        else
        {
            cout << "资源分配失败!!" << endl;
            cout << "继续操作吗？[Y/N]:";
            cin >> flag;
        }

        if (flag == "Y" || flag == "y")
        {
            ;
        }
        else
        {
            break;
        }
        BankAlgorithm(); //执行银行家算法

    }
    return 0;
```

}

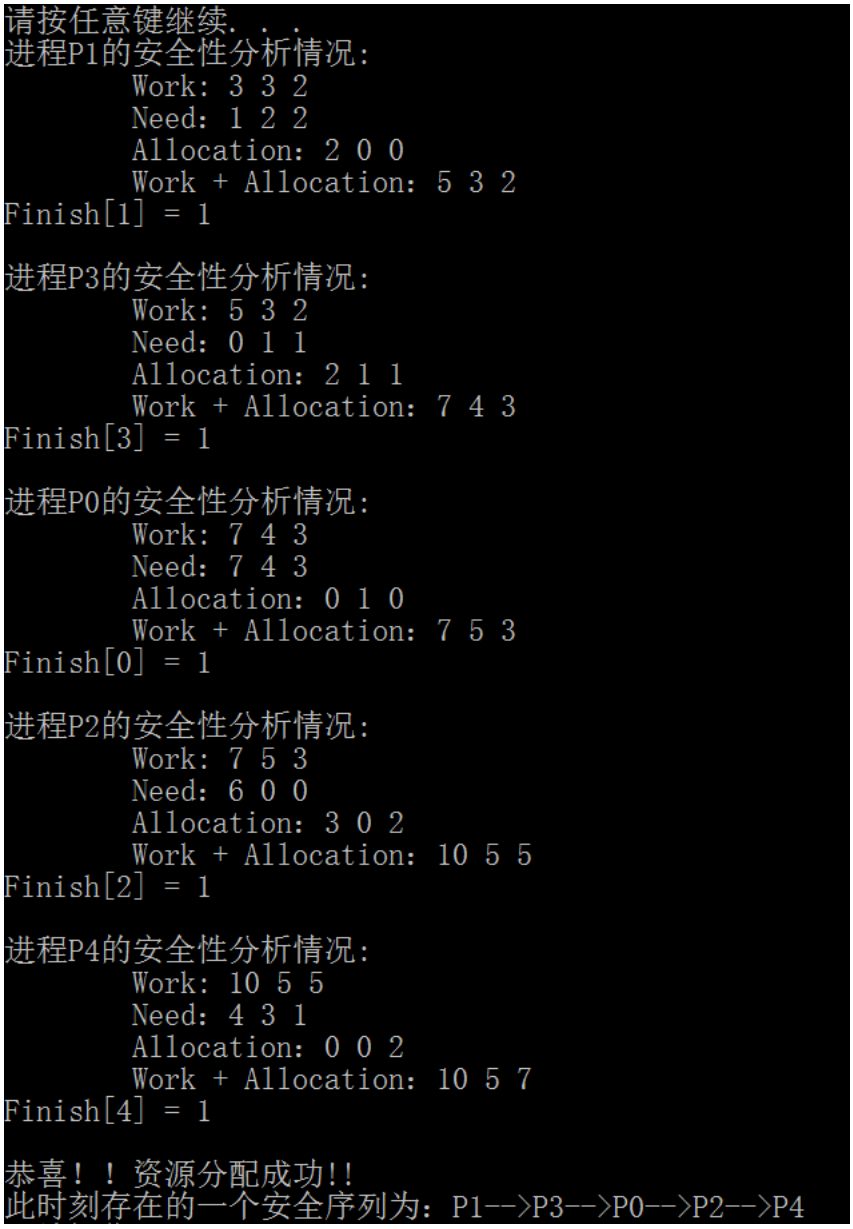
二、程序运行截图

（1）确定系统在 T0 时刻的安全性。

选择C:\WINDOWS\system32\cmd.exe

```
================================
|    姓名：田松岩              |
|    班级：2015级4班           |
|    学号：2015015335          |
|    任务：银行家算法的模拟实现 |
================================
请输入系统所要运行的进程总数:5
请输入系统中分配的资源种类数：3
请分别输入3类资源的当前可利用资源数目:3 3 2
请输入各进程对各资源的最大需求矩阵(按5*3矩阵格式输入):
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
请输入分配矩阵(按5*3矩阵格式输入):
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
请输入此刻的需求矩阵(按5*3矩阵格式输入):
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
```

```
进程P0的总体分配情况:
        Max: 7 5 3
        Allocation: 0 1 0
        Need: 7 4 3

进程P1的总体分配情况:
        Max: 3 2 2
        Allocation: 2 0 0
        Need: 1 2 2

进程P2的总体分配情况:
        Max: 9 0 2
        Allocation: 3 0 2
        Need: 6 0 0

进程P3的总体分配情况:
        Max: 2 2 2
        Allocation: 2 1 1
        Need: 0 1 1

进程P4的总体分配情况:
        Max: 4 3 3
        Allocation: 0 0 2
        Need: 4 3 1

可利用资源情况: 3 3 2

请按任意键继续. . .
```



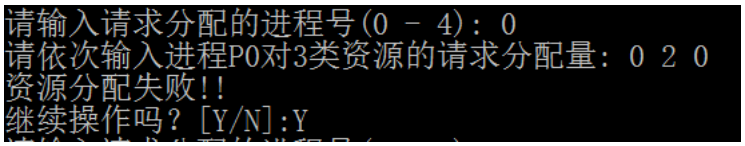
（2）P1 发出资源请求向量 Request1(1,0,2)，按照银行家算法确定能否将资源分配给 P1。

```
继续操作吗？[Y/N]:Y
请输入请求分配的进程号(0 - 4): 1
请依次输入进程P1对3类资源的请求分配量: 1 0 2
进程P1的安全性分析情况:
        Work: 2 3 0
        Need: 0 2 0
        Allocation: 3 0 2
        Work + Allocation: 5 3 2
Finish[1] = 1

进程P3的安全性分析情况:
        Work: 5 3 2
        Need: 0 1 1
        Allocation: 2 1 1
        Work + Allocation: 7 4 3
Finish[3] = 1

进程P0的安全性分析情况:
        Work: 7 4 3
        Need: 7 4 3
        Allocation: 0 1 0
        Work + Allocation: 7 5 3
Finish[0] = 1

进程P2的安全性分析情况:
        Work: 7 5 3
        Need: 6 0 0
        Allocation: 3 0 2
        Work + Allocation: 10 5 5
Finish[2] = 1

进程P4的安全性分析情况:
        Work: 10 5 5
        Need: 4 3 1
        Allocation: 0 0 2
        Work + Allocation: 10 5 7
Finish[4] = 1

恭喜！！资源分配成功!!
此时刻存在的一个安全序列为：P1-->P3-->P0-->P2-->P4
```

（3）在（2）的基础上，P4 发出请求向量 Request4(3,3,0)，按照银行家算法确定能否将资源分配给 P1。

```
请输入请求分配的进程号(0 - 4): 4
请依次输入进程P4对3类资源的请求分配量: 3 3 0
没有足够的资源，请求资源失败!!
恭喜！！资源分配成功!!
此时刻存在的一个安全序列为：P1-->P3-->P0-->P2-->P4
```

（4）再（3）的基础上，P0 发出请求向量 Request0(0,2,0)，按照银行家算法确定能否将资源分配给 P1。



三、收获、体会及对该实验的改进意见和见解

收获：深刻的理解了银行家算法、系统安全状态，掌握了死锁的原因、条件、处理方法

学会了用 processon 制作流程图，回忆了上学期学的 C++

实验还有很多不够严谨的地方，我会随着不断的学习，有了更多的体会和了解后在不断的完善。