吉林大学

软件学院

操作系统课程设计实验报告

班级	6		
学号	55210629		
姓名	张耕儒		

题目：**基于死锁避免动态策略的资源分配银行家算法的模拟实现**

一、实验内容

(1)设计银行家算法所需的数据结构设置（界面）

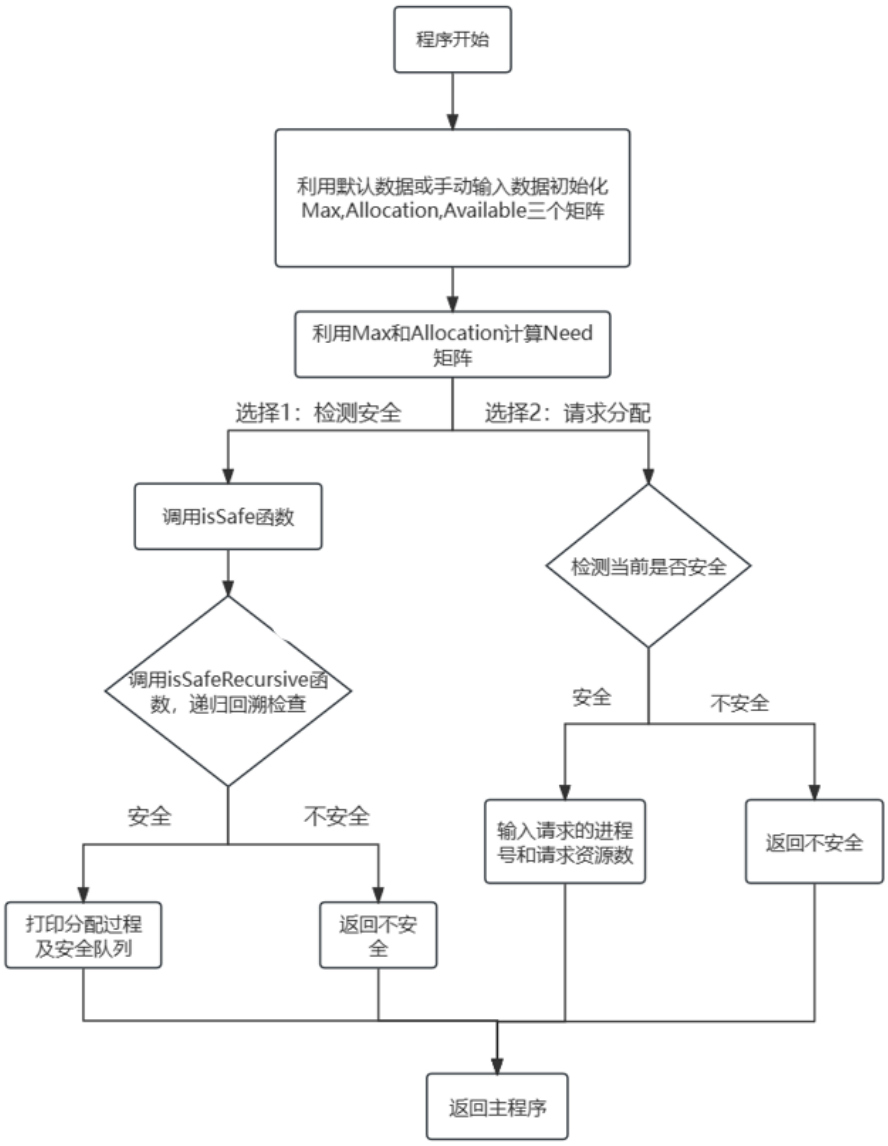
(2)运行结果（界面）

- ① 显示当前所设置的数据状态下，系统是否是安全状态的提示信息；
- ② 如果系统是安全状态，要求给出此状态下所有的安全进程序列；
- ③ 模拟在当前状态下，给出任意进程对某资源类的申请，并按银行家显示是否可接受的提示信息；如果该申请是可接受的，给出系统接受申请后的系统资源情况

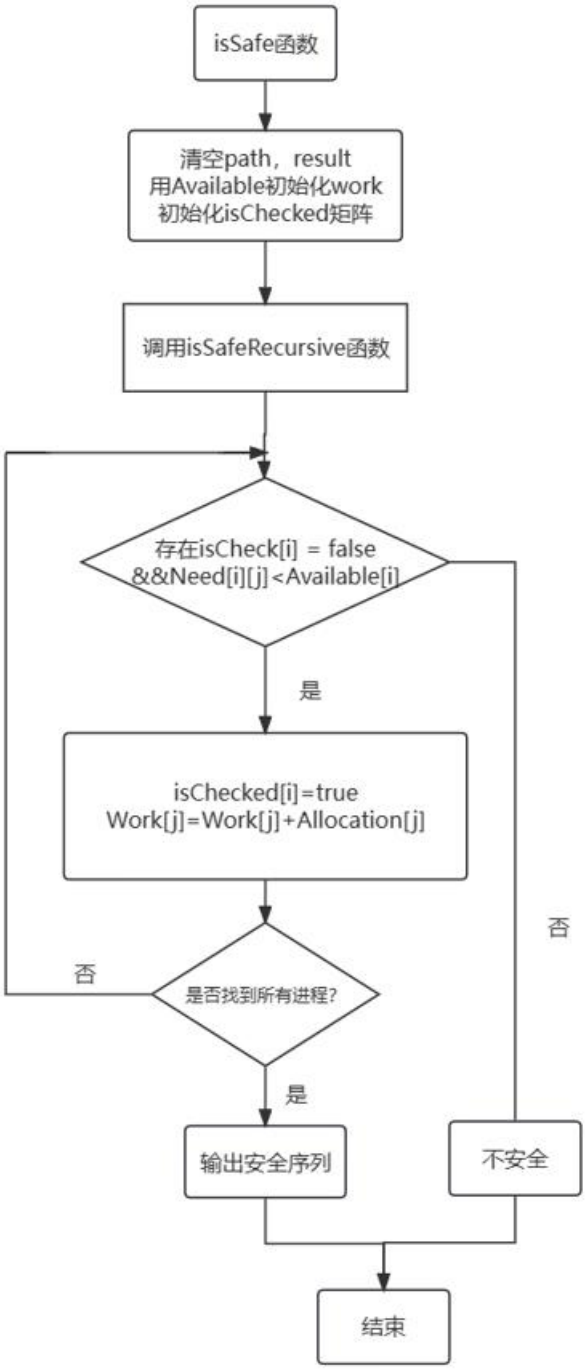
张耕儒完成（1），（2）①，②，③

二、实验设计

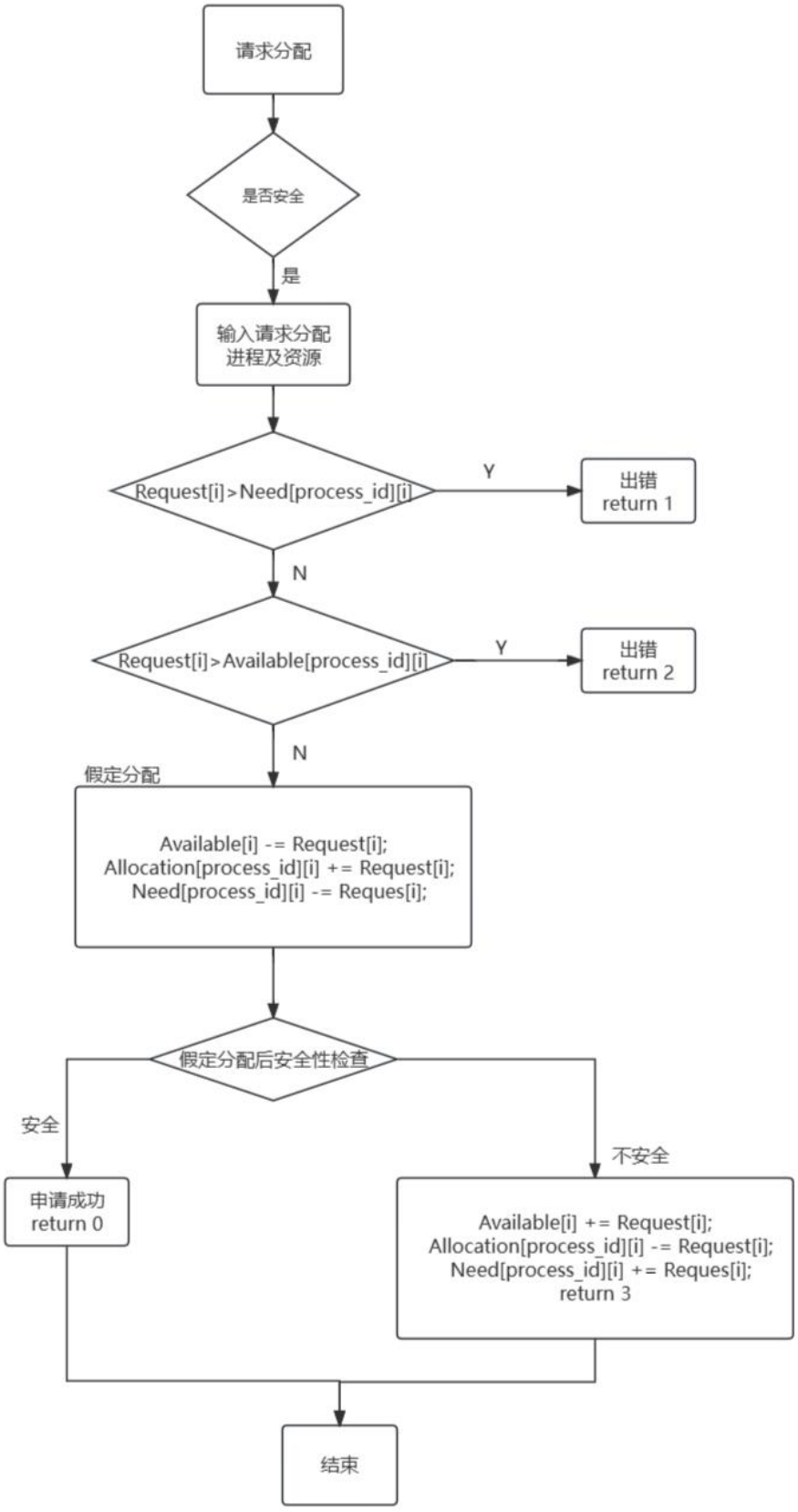
1. 主程序：



2.安全性检查：


isSafe函数
清空path，result
用Available初始化work
初始化isChecked矩阵
调用isSafeRecursive函数
存在isCheck[i] = false
&&Need[i][j]<Available[i]
是
isChecked[i]=true
Work[j]=Work[j]+Allocation[j]
否
是否找到所有进程?
否
是
输出安全序列
不安全
结束

3.请求资源


请求分配
是否安全
是
输入请求分配
进程及资源
Request[i]>Need[process_id][i]
Y
出错
return 1
N
Request[i]>Available[process_id][i]
Y
出错
return 2
N
假定分配
Available[i] -= Request[i];
Allocation[process_id][i] += Request[i];
Need[process_id][i] -= Reques[i];
假定分配后安全性检查
安全
不安全
申请成功
return 0
Available[i] += Request[i];
Allocation[process_id][i] -= Request[i];
Need[process_id][i] += Reques[i];
return 3
结束

三、编码实现

检测是否安全代码：

```
bool Banker::isSafe()
{
    vector<int> work;
    work = Available;
    vector<bool> isChecked(processNum, false);
    isSafeRecursive(work, isChecked);
    if (result.empty())
        return false;
    return true;
}
void Banker::isSafeRecursive(vector<int>& work, vector<bool>& isChecked)
{
    bool hasFound = false;
    if (path.size() == processNum)
    {
        result.push_back(path);
        safePrint(print_num);
        add_printNum();
        return;
    }
    for (int i = 0; i < processNum; i++)
    {
        if (!isChecked[i])
        {
            for (int j = 0; j < resourceNum; j++)
            {
                if (Need[i][j] > work[j])
                    break;
                if (j == resourceNum - 1)
                {
                    for (int k = 0; k < resourceNum; k++)
                        work[k] += Allocation[i][k];
                    isChecked[i] = true;
                    hasFound = true;
                    path.push_back(i);
                    resourcePrint(work, i, isChecked);
                    isSafeRecursive(work, isChecked);
                    path.pop_back();
                    isChecked[i] = false;
                    for (int k = 0; k < resourceNum; k++)
                        work[k] -= Allocation[i][k];
                }
```

```
            }
        }
    }
    if (!hasFound)
        return;
}
```

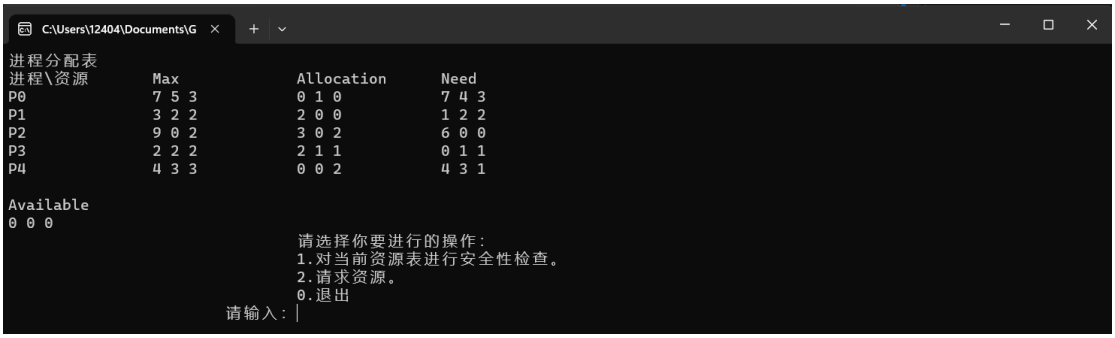
分配资源代码：

```
int Banker::requestResource(int process_id, vector<int> req)
{
    cout << "开始分配：" << endl << endl;
    for (int i = 0; i < resourceNum; i++)
        if (req[i] > Need[process_id][i])
            return 1;
    for (int i = 0; i < resourceNum; i++)
        if (req[i] > Available[i])
            return 2;
    for (int i = 0; i < resourceNum; i++)   //分配资源
    {
        Available[i] -= req[i];
        Allocation[process_id][i] += req[i];
        Need[process_id][i] -= req[i];
    }
    resourcePrint();
    if (!isSafe())
    {
        for (int i = 0; i < resourceNum; i++)   //回收资源
        {
            Available[i] += req[i];
            Allocation[process_id][i] -= req[i];
            Need[process_id][i] += req[i];
        }
        return 3;
    }
    else
        for (int i = 0; i < resourceNum; i++)   //恢复
        {
            Available[i] += req[i];
            Allocation[process_id][i] -= req[i];
            Need[process_id][i] += req[i];
        }
    return 0;
}
```

```
int Banker::requestResource(int process_id, vector<int> req)
{
    cout << "开始分配：" << endl << endl;
    for (int i = 0; i < resourceNum; i++)
        if (req[i] > Need[process_id][i])
            return 1;
    for (int i = 0; i < resourceNum; i++)
        if (req[i] > Available[i])
            return 2;
    for (int i = 0; i < resourceNum; i++)   //分配资源
    {
        Available[i] -= req[i];
        Allocation[process_id][i] += req[i];
        Need[process_id][i] -= req[i];
    }
    resourcePrint();
    if (!isSafe())
    {
        for (int i = 0; i < resourceNum; i++)   //回收资源
        {
            Available[i] += req[i];
            Allocation[process_id][i] -= req[i];
            Need[process_id][i] += req[i];
        }
        return 3;
    }
    //下属代码看需求，如果是连续分配，则不需要恢复，如果只是尝试单次，即可恢复
    /*else
        for (int i = 0; i < resourceNum; i++)   //恢复
        {
            Available[i] += req[i];
            Allocation[process_id][i] -= req[i];
            Need[process_id][i] += req[i];
        }
    return 0;*/
}
```

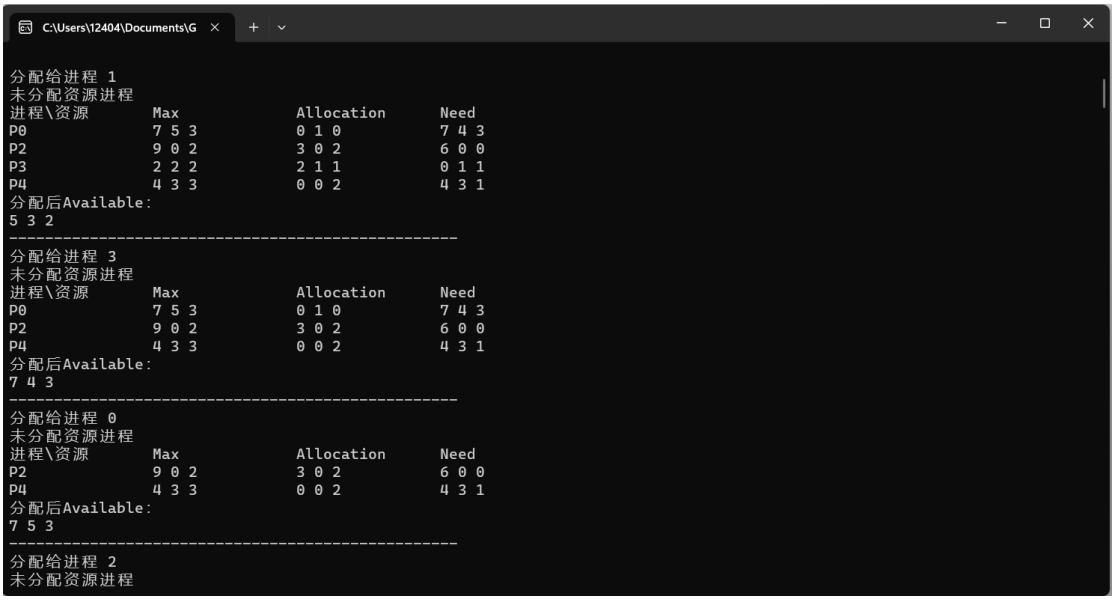
四、实验结果

1.程序开始：



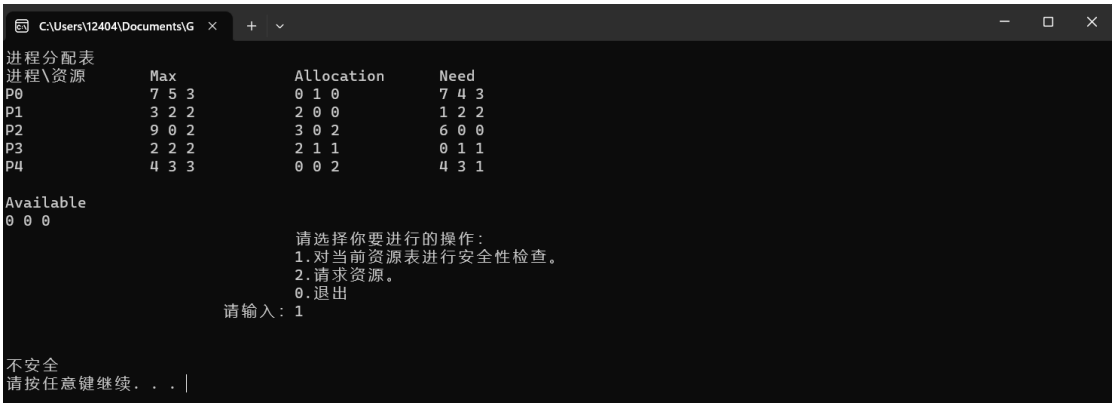
2.安全性检查

①分配过程



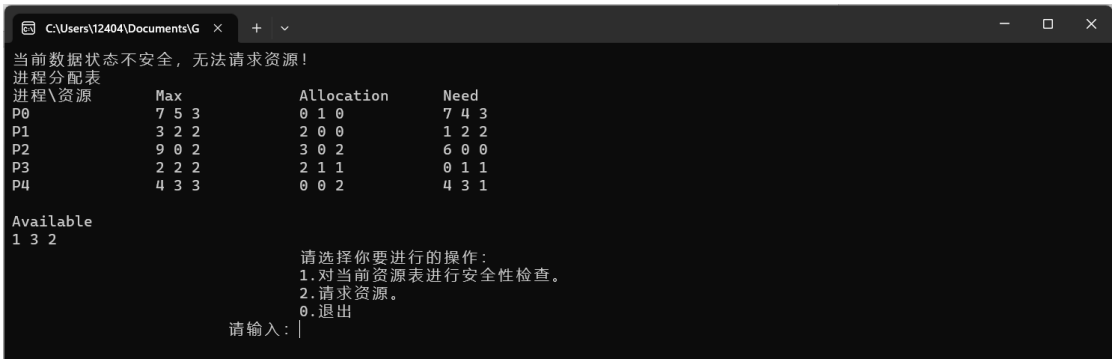
②所有安全序列

③不安全状态



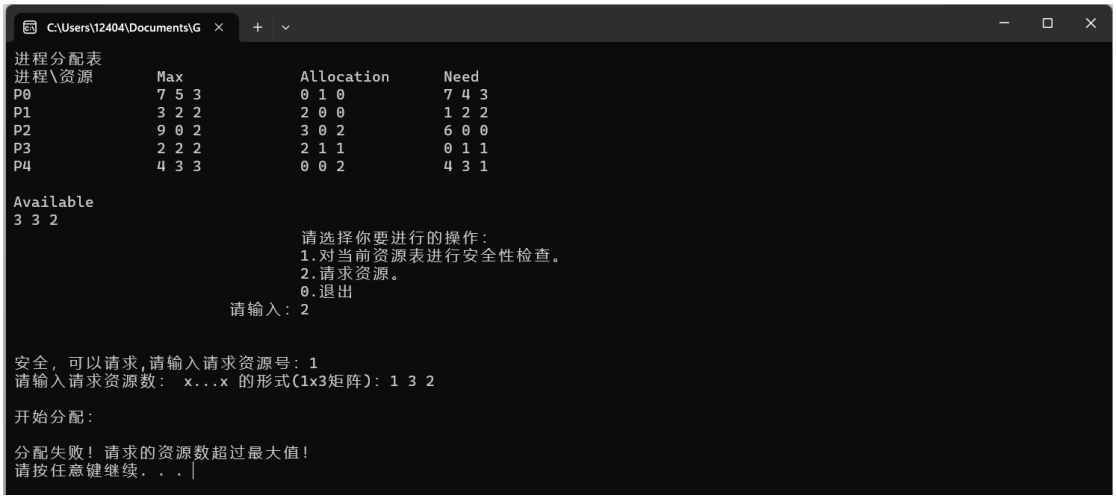
3. 请求资源：

①不安全状态

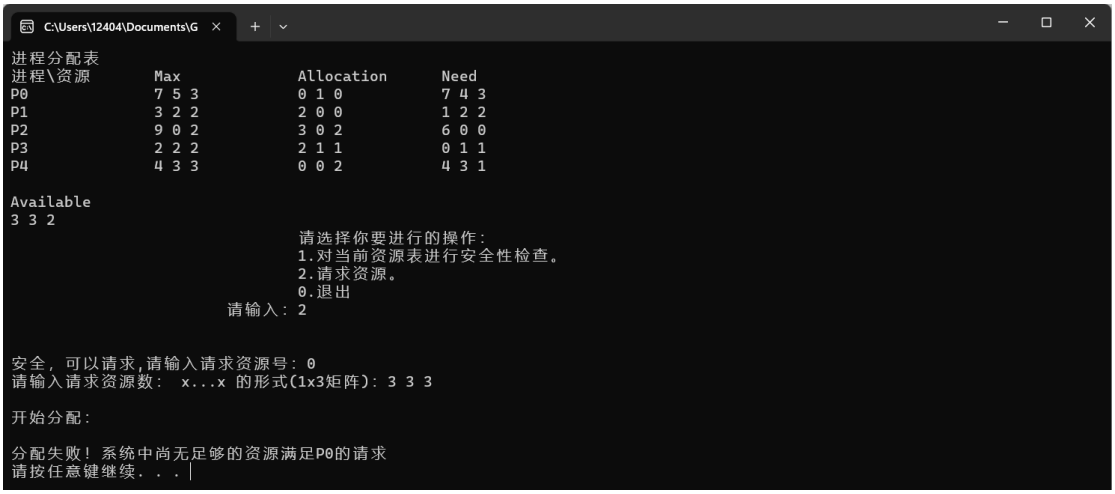


②安全

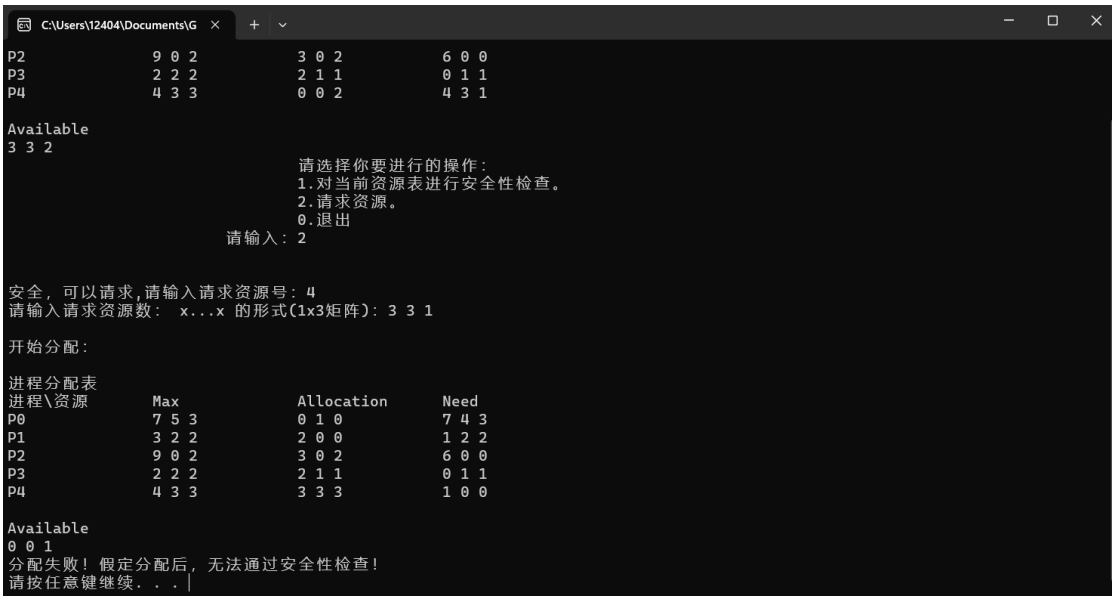
Ⅰ.请求资源超过最大请求值



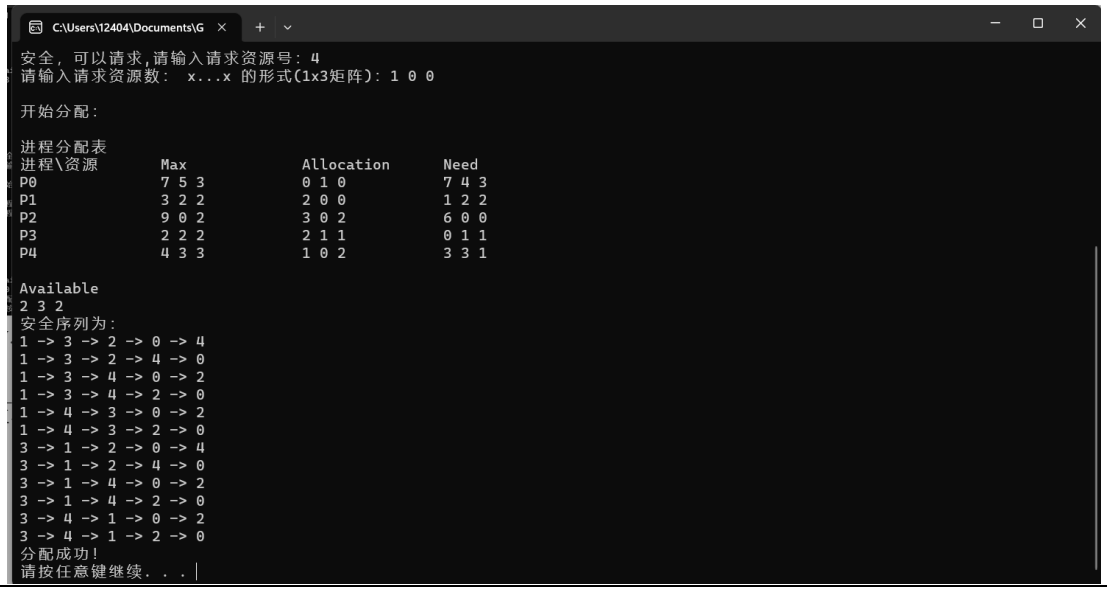
II. Available 小于请求资源



III. 假定分配后，无法通过安全性检查



IV. 分配成功



五、思考与体会

这次课程设计实现了操作系统第五次实验银行家算法，让我更加深刻的理解了进程死锁这一概念，也学习了如何进行死锁预防和死锁避免，而操作系统课程设计银行家算法正是通过死锁避免来实现的，通过自己编写代码，大大加深了对于操作系统第五次实验银行家算法细节的理解，理解了如何通过安全性算法来检查系统状态是否安全，也更加理解了如何通过资源请求算法来请求资源分配和释放未成功分配的资源。