

**《操作系统课程设计》报告**

**设计题目 编程模拟银行家算法**

**学院名称 计算机与网络安全学院（牛津布鲁克斯学院）**

**专业名称 软件工程**

**学生姓名 郑昌良**

**学生学号 202013160217**

**指导教师 王宇**

**设计成绩**

**教务处 制**

2022年09月16日

目录

[一、设计目的 1](#_Toc114326368)

[二、设计内容 1](#_Toc114326369)

[三、背景信息 1](#_Toc114326370)

[3-1 死锁的概念 1](#_Toc114326371)

[3-2 银行家算法的由来 2](#_Toc114326372)

[3-3 银行家算法的作用 3](#_Toc114326373)

[四、银行家算法的原理及实现过程 3](#_Toc114326374)

[4-1 主程序工作流程 3](#_Toc114326375)

[4-2 安全性算法 4](#_Toc114326376)

[4-3 银行家算法 6](#_Toc114326377)

[4-4 一些功能性函数 8](#_Toc114326378)

[五、测试及设计结果 10](#_Toc114326379)

**模拟银行家算法**

## 设计目的

通过设计和调试银行家算法通用程序，加深对死锁概念和死锁避免方法的了解。

## 设计内容

编制银行家算法程序，并检测所给状态的系统安全性。

## 背景信息

### 3-1 死锁的概念

（1）定义：指的是多个进程在运行过程中因为争夺资源而造成的种僵局，当进程处于这种僵局状态时，若无外力作用，他们都将无法再向前推进的状态。

（2）产生死锁的原因：竞争非可剥夺性型资源、进程推进不当。

（3）产生死锁的必要条件

互斥条件；

请求和保持条件；

不可剥夺条件；

环路等待条件。

（4）处理死锁的基本方法

预防死锁: 属于事前预防的策略，通过设置某些限制条件，去破坏产生死锁的四个必要条件或其中的几个条件。预防死锁比较容易实现，所以被泛使用，但是由于施加的限制条件过于严格可能会导致系统资源利用率和系统吞吐量降低。

避免死锁: 属于事前预防的策略，但它并不需要事先采取各种限制措施去破坏产生死锁的四个必要条件，而是在资源的动态分配过程中，用某种方法去防止系统进入不安全状态，从而避免死锁的产生。但实现有一定的难度。目前较完善的系统中常用此法来避免死锁。

检测死锁: 这种方法不需要事前采取任何限制措施，也不用检查是否进入不安全状态，而是允许系统在运行的过程中发生死锁。但是通过系统所设置的检测机构.及时的检测出死锁的发生，并精确的测出与死锁有关的进程和资源，然后，采取适当的措施，从系统中将已发生的死锁清楚掉。

解除死锁: 这是与检测死锁相配套的一套措施。当检测到系统已经产生死锁时，须将进程从死锁中解放出来。通常用到的实施方法是撤销或挂起些进程，以便收回一些资源，再将这些资源分配给已处于阻塞状态的进程，使之转为就绪状态，以继续运行。死锁的检测和解除措施，有可能使系统获得较好的资源和吞吐量，但在现实上难度也最大。

预防死锁和避免死锁的区别: 预防死锁和避免死锁实质上都是通过施加某种相知条件的方法，来预防发生死锁。两者的主要区别:为了预防死锁所施加的限制条件较为严格，这往往会影响到进程的并发执行，而避免死锁所施加的限制条件则较为宽松，有利于进程的并发执行。

### 3-2 银行家算法的由来

Dijkstra的银行家算法是操作系统中最有代表性的死锁避免方案，该算法由于能够用于银行系统现金的发放而得名。在银行中，客户申请贷款的数量是有限的，每个客户在第一次申请贷款时要声明完成该项目所需的最大资金量，在满足所有贷款要求时，客户应及时归还。银行家在客户申请的贷款数量不超过自己拥有的最大值时，都应尽量满足客户的需要。在这样的描述中，银行家就好比操作系统，资金就是资源，客户就相当于要申请资源的进程。

操作系统作为裸机上安装的第一层软件， 起着控制和管理计算机内部软硬件资源，合理组织计算机工作流程，提高计算机工作效率，用户和计算机硬件接口的重要作用。因此操作系统即要保证系统资源的合理分配提高系统资源利用率，同时又要避免死锁等不安全状况的出现，如果这些不安全状况出现操作系统还要解决这些问题，让系统回到安全状态。银行家算法就是在这样的背景下应运而生的。

### 3-3 银行家算法的作用

银行家算法是一种最有代表性的避免死锁的算法。在避免死锁方法中允许进程动态地申请资源，但系统在进行资源分配之前，应先计算此次分配资源的安全性，若分配不会导致系统进入不安全状态，则分配，否则等待。为实现银行家算法，系统必须设置若干数据结构。要解释银行家算法，必须先解释操作系统安全状态和不安全状态。

安全序列是指一个进程序列{P1，…，Pn}是安全的，即对于每一个进程

Pi(1≤i≤n），它以后尚需要的资源量不超过系统当前剩余资源量与所有进程

Pj (j < i )当前占有资源量之和。

银行家算法的核心它通过自己特有的算法，在每次奉陪给进程系统资源前，先试探性的“假设”分配资源给进程Pi, 再通过安全性算法检测此次分配是否会导致系统进入不安全状态，如果分配后系统依然安全则系统将资源正是分配给进程Pi; 如果此次分配导致系统进入不安全状态，则暂不分配资源给进程Pi。通过这种机制，系统可以有效的避免死锁的产生，确保系统时时刻刻都处在安全状态。

## 银行家算法的原理及实现过程

### 4-1 主程序工作流程

在银行家算法正式开始运行之前，应该对系统所拥有的各项资源进行初始化。

初始化完成后，应该对当前模拟系统的安全性进行检查，如果不安全，则中断；若系统此时安全，则进行下面的操作。

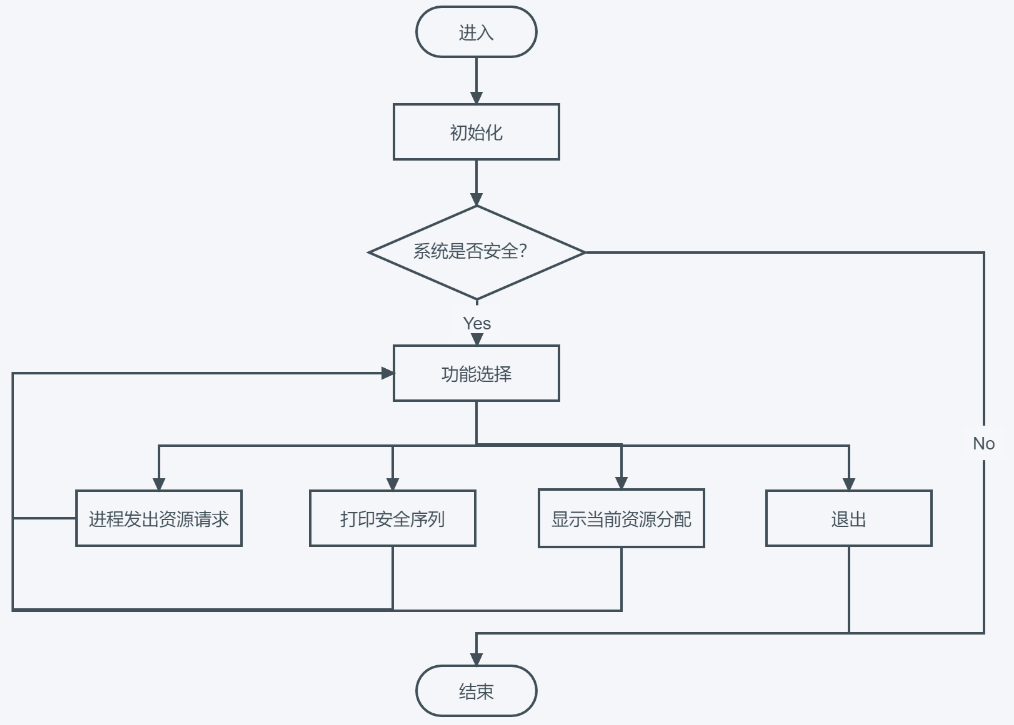


图4.1.1 主程序工作流程图

### 4-2 安全性算法

（1）安全性算法的数据结构

// 保存安全性, true为安全

bool status = false;

// 保存向量，在检查安全性时使用

vector<int> Check(Available);

// 记录该进程是否分配

vector<bool> isDistribute(process\_num, false);

// 标记不满足安全状态的进程, 每找到一个，flag--

int flag, temp\_flag;

（2）安全性算法工作流程

1. 从进程集合中找到满足条件的进程。
2. 当该进程获得资源并顺利执行过后，释放分配给该进程的资源。
3. 重复执行上述步骤，直到所有的进程的 isDistribute 都为 true后，则该系统处于安全状态，反之处于不安全状态。
4. 若该进程安全，则将该进程 id 放入安全序列。

（3）安全性算法工作流程图

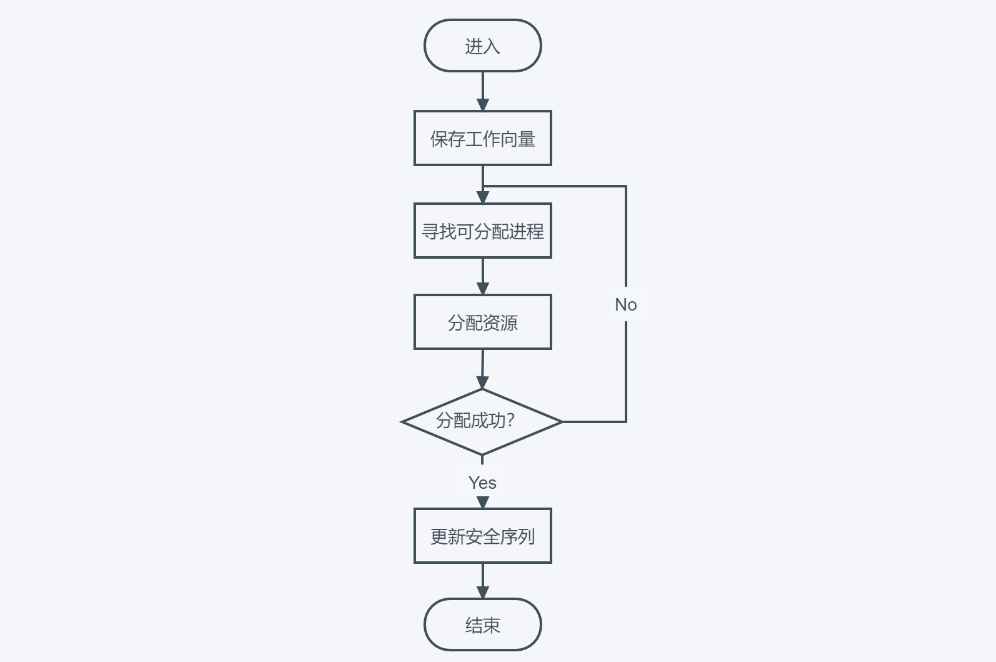


图4.2.1 安全性算法工作流程图

（4）安全性算法核心代码

while (true) {

// 统计isDistribute中所有false的个数

flag = count(isDistribute.begin(), isDistribute.end(), false);

temp\_flag = flag; // 将flag暂存temp\_flag

for (int i = 0; i < process\_num; i++) {

// 可分配，但未分配

if (Need[i] <= Check && isDistribute[i] == false) {

for (int j = 0; j < Check.size(); j++) {

// 分配过后回收资源，将资源加入Check

Check[j] += Allocation[i][j];

}

isDistribute[i] = true;

flag--;// 找到一个，flag--

        securityList.push\_back(i); // 将进程 id 放入安全序列

}

}

// 判断一下flag，以确定系统是否安全

// 当所有的进程状态均为安全，则系统处于安全状态

if (flag == 0) {

status = true;break;}

else if (temp\_flag == flag) {

status = false;break;}

}

### 4-3 银行家算法

（1）银行家算法数据结构

// 最大资源数

int max\_size = 6;

// 进程数量

int process\_num;

// 各类资源数

int resourse\_num;

// 进程所需要的资源总量

vector<vector<int>> Claim;

// 已经分配给进程的资源数

vector<vector<int>> Allocation;

// 进程还需要的资源

vector<vector<int>> Need;

// 系统剩余的可用资源

vector<int> Available;

// 安全序列，存储进程 id

vector<int> securityList;

// 资源分配状态

string distribute\_status;

（2）银行家算法工作流程

1. 进入算法时先检查本次资源请求是否合理，检查项主要包括请求量是否超过库存量，当前的系统安全性。

2. 检查资源请求合理后，对该进程进行资源预分配。预分配完成后再次检查系统当前安全性。

3. 若当前系统安全性为不安全，则进行回退，将刚刚预分配的资源收回。

4. 若当前系统安全性为安全，则正式分配资源。待进程运行结束后，收回分配的资源。

（3）银行家算法工作流程图

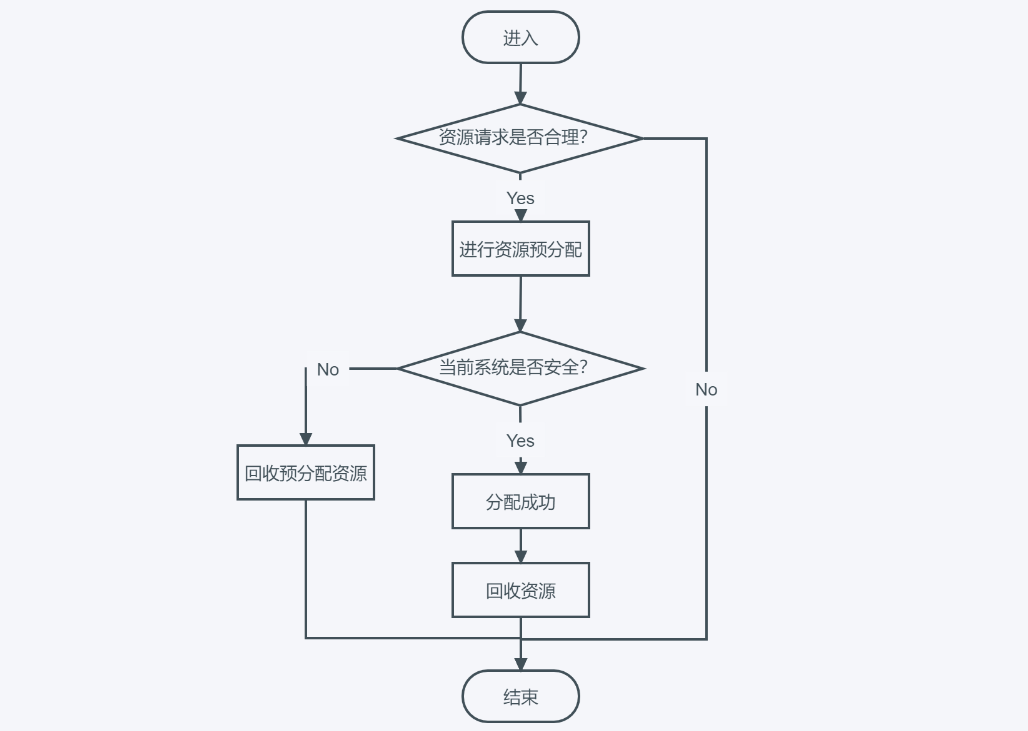


图4.3.1 银行家算法工作流程图

（4）银行家算法核心代码

* 资源预分配

for (int i = 0; i < resourse\_num; i++) {

   // 将系统可用资源减去请求资源

   Available[i] -= req[i];

   // 将已经分配资源矩阵加上请求资源

   Allocation[process\_id][i] += req[i];

   // 将还需要资源的矩阵减去请求矩阵

   Need[process\_id][i] -= req[i];}

* 回收资源

// 这里直接比较会出错，重载运算符 ==，见末尾功能性函数段

cout << endl << "当前系统处于安全状态... " << endl;

if (Need[process\_id] == 0) {

for (int i = 0; i < process\_num; i++) {

   // 将已分配矩阵的资源还给可用资源矩阵

   Available[i] += Allocation[process\_id][i];

   // 该进程的已分配资源清零

   Allocation[process\_id][i] -= Allocation[process\_id][i];

   }

}

* 回退（预分配的逆操作）

for (int i = 0; i < process\_num; i++) {

   // 将暂定分配资源归还，做预分配的逆操作

   Available[i] += req[i];

   Allocation[process\_id][i] -= req[i];

   Need[process\_id][i] += req[i];

}

distribute\_status="分配失败！此次分配会导致系统处于不安全状态，已挂起！";

### 4-4 一些功能性函数

* 重载运算符 <=

在银行家算法中，会出现两个动态数组比较大小的情况，这里考虑到如果银行家算法中再增添比较两个数组的代码，会使关键代码变得复杂，这里将 <= 运算符重载写在外边。

bool operator<=(vector<int> &req, vector<int> &need) {

    for (int i = 0; i < req.size(); i++) {

        if (req[i] > need[i]) {

            return false;

        }

    }

    return true;

}

* 重载运算符 ==

在银行家算法中，判断进程是否还需要资源的时候，会将该进程需求矩阵中的每一类资源的需求量与 0 比较，同上，将重载写在外边。

bool operator==(vector<int> &need, int n) {

    for (auto &e : need) { //与for(;;)效果相同

        if (e != n) return false;

    }

    return true;

}

* 重载运算符 <<

在打印矩阵时，如果每个矩阵都单独写一个循环的话，会使代码变得冗长，这里将输出流函数重载，实现输出一维数组和二维数组。

// 输出一维数组

ostream &operator<<(ostream &out, vector<int> &A) {

    out << "[";

    for (int i = 0; i < A.size(); i++) {

        out << A[i] << ",";

    }

    out << "\b"<< "]";

    return out;

}

// 输出二维数组

ostream &operator<<(ostream &out, vector<vector<int>> &A) {

    out << endl;

    for (int i = 0; i < A.size(); i++) {

        out << "[";

        for (int j = 0; j < A[i].size(); j++) {

            out << A[i][j] << ",";

        }

        out << "\b"

            << "]" << endl; //输出完一行过后换行

    }

    return out;

}

1. 打印资源分配表

* 打印表格

在打印资源分配表时，期望将数据更直观地展示给用户，需要对输出数据进行简单的排版。

void printTable() {

    cout << "==========================资源表格=================="

            "========"<< endl;

    for (int i = 0; i < process\_num; i++) {

        cout << "P" << i;

        cout << "  Claim:[";

        for (int j = 0; j < resourse\_num; j++) {

            cout << setw(2) << Claim[i][j] << ",";

        }

        cout << "\b"<< "]";

        cout << "  Allocation:[";

        for (int j = 0; j < resourse\_num; j++) {

            cout << setw(2) << Allocation[i][j] << ",";

        }

        cout << "\b"<< "]";

        cout << "  Need:[";

        for (int j = 0; j < resourse\_num; j++) {

            cout << setw(2) << Need[i][j] << ",";

        }

        cout << "\b"<< "]" << endl;

    }

    cout << "\n当前系统可用资源 Available: " << Available << endl;

}

## 测试及设计结果

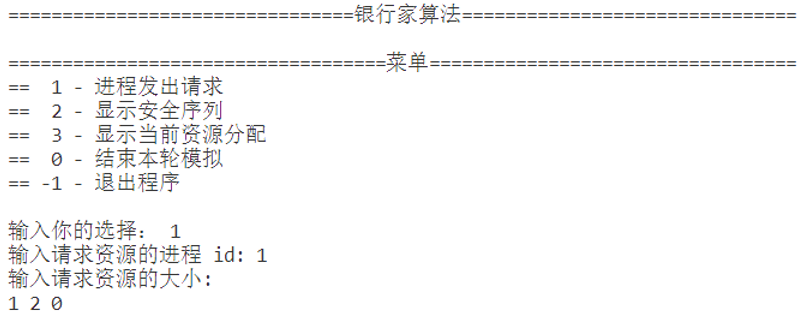
1. 测试用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Allocation** | | | **Need** | | |
| **0** | **1** | **0** | **6** | **4** | **3** |
| **2** | **0** | **0** | **1** | **2** | **0** |
| **3** | **0** | **2** | **6** | **0** | **0** |
| **2** | **1** | **1** | **0** | **1** | **1** |
| **0** | **0** | **2** | **4** | **3** | **1** |
| **Available** | | |  |  |  |
| **3** | **3** | **2** |  |  |  |

表5.1 测试用例

（2）测试结果

* 进程 P1 请求资源（1，2，0），测试结果：“分配成功！”。

图5.1 菜单

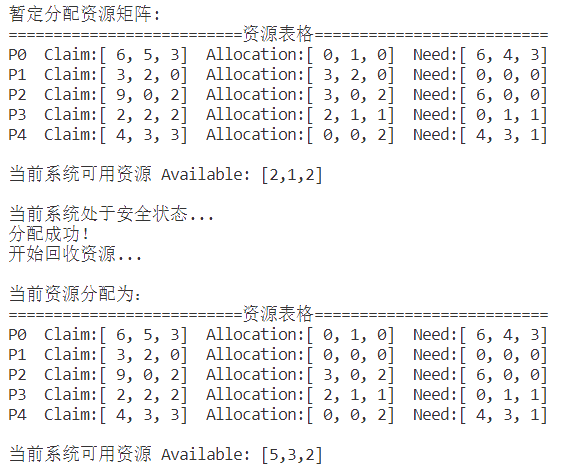


图5.2 分配成功示例

* 进程 P1 请求资源（3，2，1），测试结果：“请求资源大于所需要的资源！”。

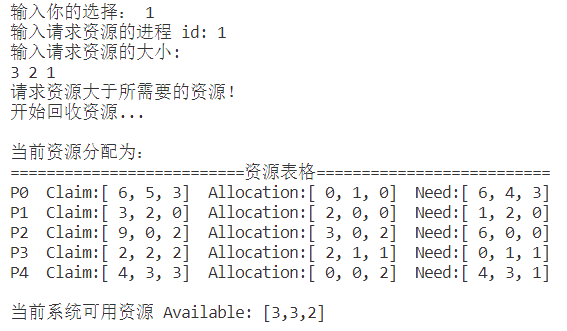


图5.3 请求资源大于所需资源

* 进程 P1 请求资源（7，6，5），测试结果：“请求资源大于系统拥有的资源！”。

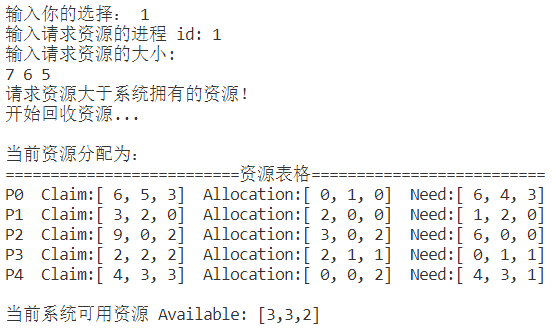


图5.4 请求资源大于系统资源

* 进程 P0 请求资源（3，3，2），测试结果：“分配失败！此次分配会导致系统处于不安全状态，已挂起！”。

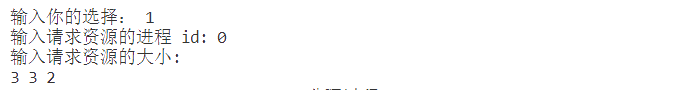


图5.5 分配失败示例-1

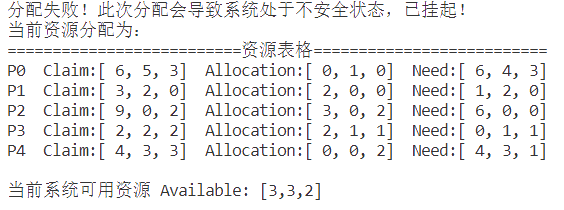


图5.6 分配失败示例-2

以上测试用例覆盖了“分配失败”、“请求资源大于系统拥有资源”、“进程还需请求更多资源”、“分配成功”，覆盖了程序能遇到的大部分问题，并且避免了死锁的问题。

* 显示安全序列

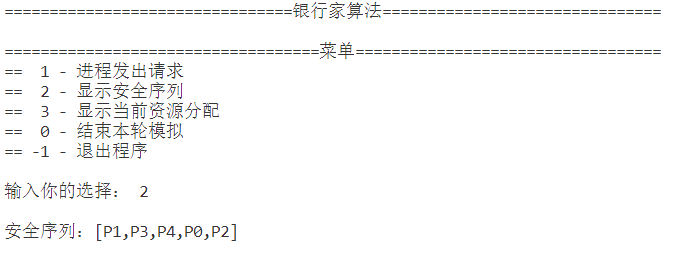


图5.7 打印安全序列

* 显示当前资源分配

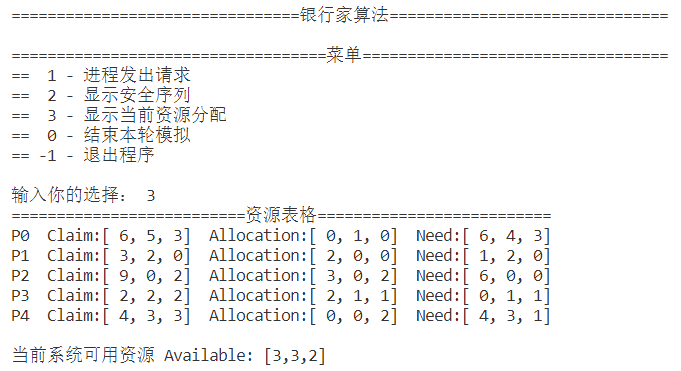


图5.8 打印资源分配表

|  |  |
| --- | --- |
| **学生实验 心得** | 通过本次课程设计的设计过程，我加深了对计算机操作系统进程管理的理解。深入的了解了银行家算法的原理和工作过程。在代码编写的过程中，也遇到了一些问题，比如已经遗忘的重载运算符等知识，但也是这个机会，让我又复习了一遍相关的知识。在编写银行家算法的过程中，加深了对死锁等概念的理解。在本次课程设计的过程中，也掌握了一些平时是可能会忽略的知识，学会了如何设计测试用例，尽量覆盖到程序运行时会遇到的各种问题。  学生（签名）：  2022年 09 月 16 日 |
| **诚信承诺** | 本人郑重声明所呈交的课程报告是本人在指导教师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注的地方外，论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同学对本文研究所做的贡献均已在报告中作了明确的说明并表示谢意。  学生（签名）：  2022年 09 月 16 日 |
| **指导**  **教师**  **评语** | 成绩评定：  指导教师（签名）：  年 月 日 |