****

**实 验 报 告**

**实验课程： 操作系统课程实践**

**学生姓名： 马星**

**学 号： 5418122020**

**专业班级： 计算机科学与技术(卓越)221班**

**2024年11月1日**

** 南昌大学实验报告**

**---（3）编程实现银行家安全算法**

学生姓名： 马星 学 号： 541812020 专业班级：计算机科学与技术(卓越)221班

实验类型：□ 验证 □ 综合 ■ 设计 □ 创新 实验日期： 11/15 实验成绩：

# 一、实验目的

通过实验加强对银行家安全算法的理解和掌握。

# 二、实验内容

熟悉避免死锁发生的方法，死锁与安全序列的关系，编程实现银行家算法，要求输出进程的安全序列。

# 三、实验要求

1、需写出设计说明；

2、设计实现代码及说明

3、运行结果；

# 四、主要实验步骤

## 1. 设计说明

银行家算法: 银行家算法是一种用于避免死锁的资源分配算法。它通过模拟资源分配和回收过程，确保系统始终处于安全状态。如果一个请求导致系统进入不安全状态，则该请求将被拒绝。

安全状态: 如果可以找到一个安全序列(按照安全序列分配资源、回收, 可以确保所有进程顺利执行完毕), 则系统属于安全状态, 否则系统处于不安全状态, 可能会发生死锁情况。

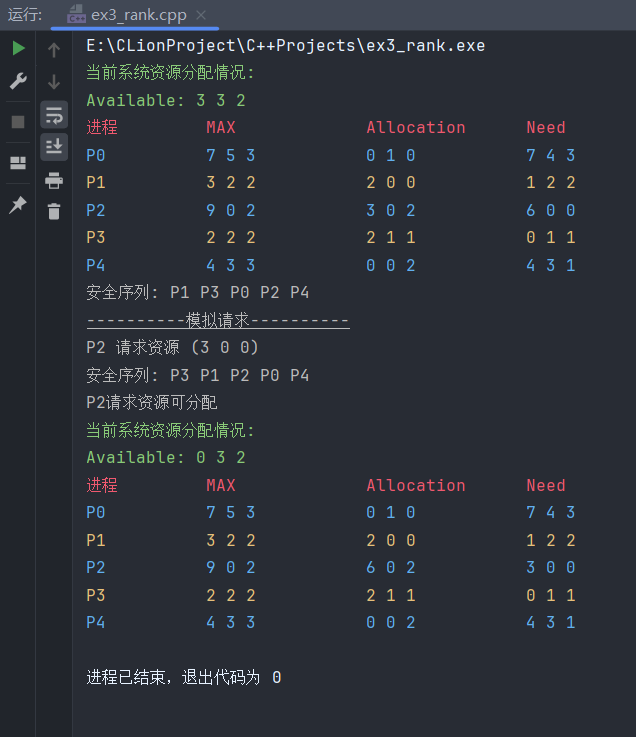
本实验中，我定义了五个进程和三种类型的资源。每个进程都有一个最大需求矩阵（Max）、已分配资源矩阵（Allocation）和需求矩阵（Need）。系统初始状态下有可用资源向量（Available）。然后模拟了进程请求资源的情况。

## 2. 实现代码

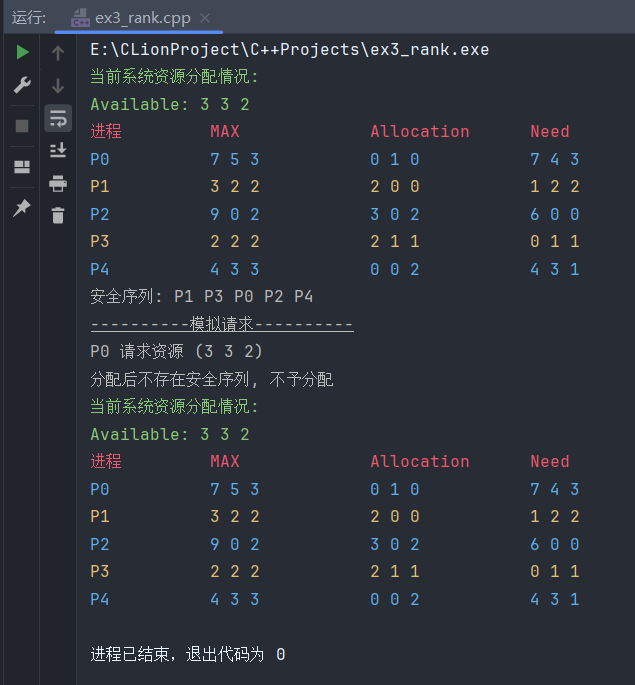
*#include* <iostream>  
*#include* <vector>  
*#include* <memory>  
*#include* "VectorPrint.h"  
*#include* "VectorUtil.h"  
*#include* "ColorPrint.h"  
  
*using namespace* std;  
*int* processCount;  
  
*/\*\*  
 \* 是否存在安全序列  
 \* @param Max 各进程的最大需求资源数, unused  
 \* @param Allocation 各进程的已分配资源数  
 \* @param Available 当前系统可用资源数  
 \* @return 不存在返回空指针, 存在返回安全序列  
 \*/  
int* \*isSafe(vector<vector<*int*>> Max, vector<vector<*int*>> Allocation, vector<vector<*int*>> Need, vector<*int*> Available) {  
 vector<*bool*> finish(processCount, *false*);  
 vector<*int*> work = Available;  
 *auto* safeSequence = make\_unique<*int*[]>(processCount);  
 *for* (*int* i = 0; i < processCount; i++) {  
 *// 查找第一个Need<work的未完成进程  
 int* j = 0;  
 *for* (; j < processCount; j++) {  
 *if* (finish[j]) *continue*;*//该进程已完成  
 if* (!isAllBiggerOrEqual(work, Need[j])) *continue*;  
 *// 分配、回收、标记为已完成* work = Add(work, Allocation[j]);  
 safeSequence[i] = j;  
 finish[j] = *true*;  
 *break*;  
 }  
 *if* (j == processCount) *return nullptr*;*// 未找到* }  
 *return* safeSequence.release();  
}  
  
*void* printCurrSysState(vector<vector<*int*>> Max, vector<vector<*int*>> Allocation, vector<vector<*int*>> Need,  
 vector<*int*> Available) {  
 setColor(GREEN);  
 cout << "当前系统资源分配情况:" << endl;  
 printVector(Available, "Available: ", " ", "\n");  
 end();  
 setColor(RED);  
 cout << "进程\t\t\t" << "MAX\t\t\t\t" << "Allocation\t\t" << "Need" << endl;  
 end();  
 *for* (*int* i = 0; i < processCount; i++) {  
 setColor(i % 2 == 0 ? BLUE : YELLOW);  
 cout << "P" << i << "\t\t\t";  
 printVector(Max[i], "", " ", "\t\t\t");  
 printVector(Allocation[i], "", " ", "\t\t\t");  
 printVector(Need[i], "", " ", "\n");  
 end();  
 }  
}  
  
  
*int* main() {  
 *// (1) 初始状态定义* vector<vector<*int*>> Max = **{**{7, 5, 3},  
 {3, 2, 2},  
 {9, 0, 2},  
 {2, 2, 2},  
 {4, 3, 3}**}**;  
 vector<vector<*int*>> Allocation = **{**{0, 1, 0},  
 {2, 0, 0},  
 {3, 0, 2},  
 {2, 1, 1},  
 {0, 0, 2}**}**;  
 vector<vector<*int*>> Need = Sub(Max, Allocation);  
 vector<*int*> Available = **{**3, 3, 2**}**;  
 processCount = Max.size();  
 printCurrSysState(Max, Allocation, Need, Available);  
  
 *// (2) 检查当状态是否安全  
 int* \*safeSequence = isSafe(Max, Allocation, Need, Available);  
 *if* (safeSequence == *nullptr*) {  
 cout << "不存在安全序列";  
 *return* 0;  
 }  
 printArr(safeSequence, processCount, "安全序列: P", " P", "\n");  
  
 *// (3) 模拟请求* colorPrintStr(UnderLine, "----------模拟请求----------\n");  
 *int* requestP = 2;  
 vector<*int*> request = **{**3, 0, 0**}**;  
 cout << "P" << requestP << " 请求资源 ";  
 printVector(request, "(", " ", ")\n");  
 *if* (!isAllBiggerOrEqual(Available, request) || !isAllBiggerOrEqual(Need[requestP], request)) {  
 cout << "该请求不可分配";  
 *return* 0;  
 }  
 *// 检查安全序列* Allocation[requestP] = Add(Allocation[requestP], request);  
 Available = Sub(Available, request);  
 Need[requestP] = Sub(Need[requestP], request);  
 safeSequence = isSafe(Max, Allocation, Need, Available);  
 *if* (safeSequence == *nullptr*) {  
 cout << "分配后不存在安全序列, 不予分配" << endl;  
 *// 回退* Allocation[requestP] = Sub(Allocation[requestP], request);  
 Available = Add(Available, request);  
 Need[requestP] = Add(Need[requestP], request);  
 } *else* {  
 printArr(safeSequence, processCount, "安全序列: P", " P", "\n");  
 cout << "P" << requestP << "请求资源可分配" << endl;  
 }  
 printCurrSysState(Max, Allocation, Need, Available);  
 *return* 0;  
}

定义了五个进程三种资源的资源需求矩阵, 在检查系统安全状态后, 模拟了对进程P2请求的响应

## 3. 运行结果



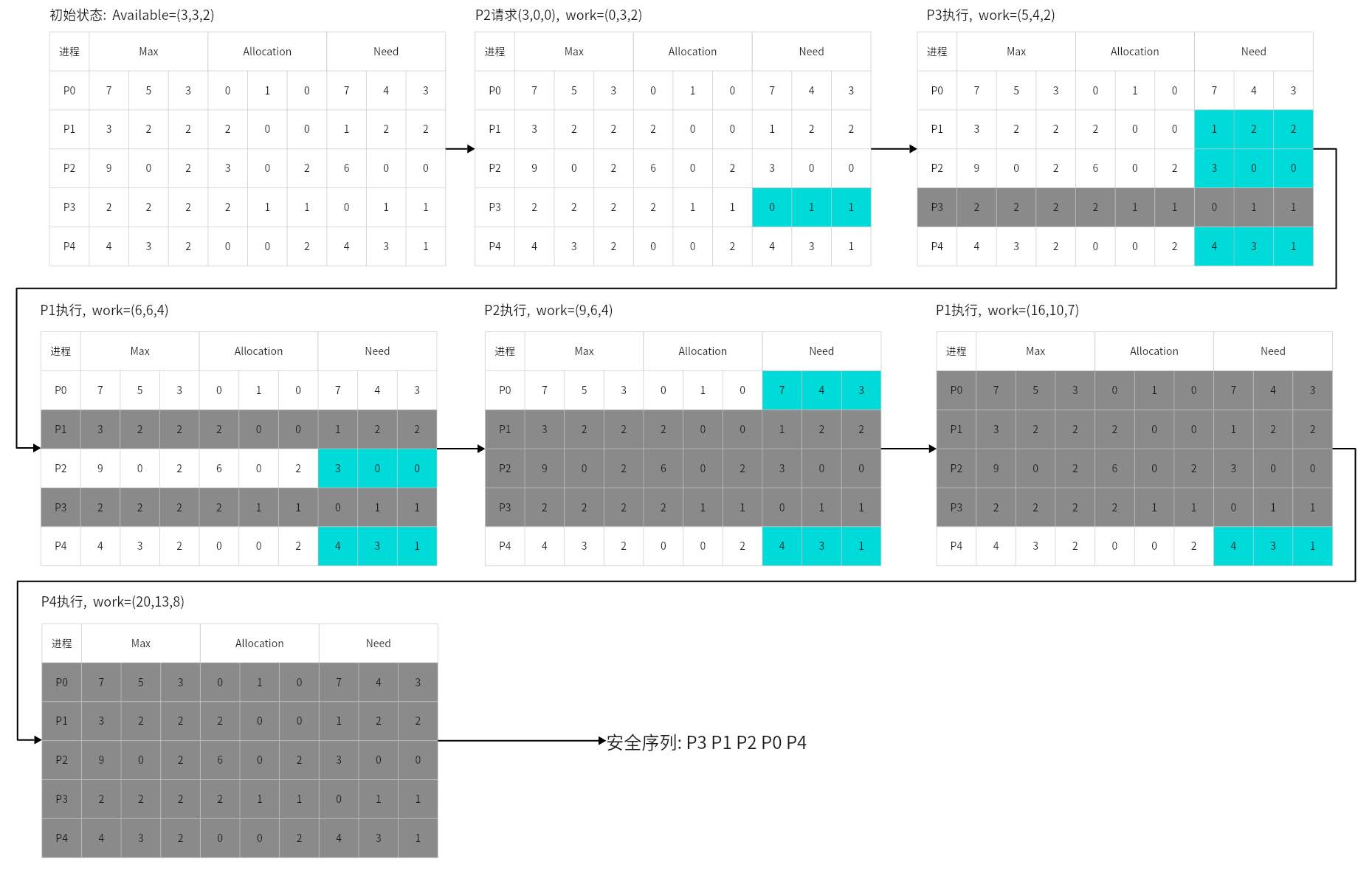
对于进程P2的请求(3,0,0), 系统找到了安全序列P3,P1,P2,P0,P4, 所以可以响应这个请求



对于进程P0的请求, 如果响应这个请求, 则找不到安全序列, 会导致系统处于不安全状态, 可能引发死锁的问题, 所以不予分配

# 五、实验数据及处理结果

手算验证P2和P1的两个请求, 检查程序是否有误



手算安全序列与程序输出一致



结果正确

# 六、实验体会或对改进实验的建议

通过本次实验，我深刻理解了银行家算法的核心思想及其在操作系统中的应用。然而，在实际使用中，银行家算法可能会面临一些挑战，例如：

* **性能问题**：随着进程和资源数量的增加，计算安全序列的时间复杂度会显著增加。可以考虑优化算法以减少计算时间。
* **动态变化**：实际系统中资源需求是动态变化的，需要实时更新资源分配表。可以引入更复杂的调度策略来应对这种动态变化。
* **多维资源**：目前实验仅考虑了单一维度的资源，实际应用中可能需要处理多维资源的情况。可以扩展算法以支持多维资源的分配。

# 七、参考资料

《计算机操作系统实验指导》