实验三 银行家算法

姓名：叶倩琳

学号：201706061330

完成日期：2019/11/12

# 一、实验目的

# 根据银行家算法的思想，编写程序，解决并发进程的死锁问题。

# 二、实验内容

# 1、进程的死锁避免算法。编写一段程序，模拟银行家算法，解决进程的死锁问题。

# 2、利用VC++6.0实现上述程序设计和调试操作，根据提示输入相应的资源请求，对于算法操作的成功与否提供一定的提示框。

# 3、通过阅读和分析实验程序，熟悉进程的死锁问题。

# 实验步骤

（1）设计算法的数据结构

考虑一个具有PNum个进程和RNum种不同类型资源的系统。

Max[MAXN][MAXN]：系统中每种资源的总量(进程i对资源 j的最大需求)

 Available[MAXN]：各资源未分配的资源数,即可用的资源数

 Allocatio[MAXN][MAXN]：Aij表示当前进程i已分配到的资源j的数量

 Need[MAXN][MAXN]：Need[i][j]=k 表示进程i还需要j类资源k个，其中Need[i][j]=Max[i][j]-Allocation[i][j]

1. 在进程并发执行时，进程i提出请求j类资源k个，即Request [i,j]=k。系统进行安全检查：
2. 如果Request [i,j]≤Need[i,j]则继续以下检查，否则显示需求申请超出最大需求值的错误。
3. 如果Request [i]≤Available则继续以下检查，否则显示系统无足够资源，Pi阻塞等待。
4. 假设同意进程i的请求，将各向量状态修改为满足请求之后的状态，然后对此状态执行安全性算法检测，判断在此次资源分配后，系统是否处于安全状态，若安全将资源分配给进程i；否则恢复原来的资源分配状态，将进程Pi置为阻塞状态。

（3）安全性算法思想

1）初始化设置工作向量Available\_copy[MAXN]表示未分配的各类资源数目，Available\_copy[i] = Available[i];设置完成标志向量Finish[n]，初始化Finish[i] = 0表示i进程尚末完成。

2）若进程i同时满足Finish[i] = 0和Need[i]≤Available\_copy[i]，继续执行；如果所有的Finish[i]＝true，则表示系统处于安全状态，否则系统处于不安全状态。

3）假设同意进程i的请求，将各向量状态修改为满足请求之后的状态，然后继续执行，并释放出分配给它的资源，Available\_copy[jj] += Allocation[i][jj];并设置Finish[i]=true ；转执行步骤2）。

# 四、主要程序结构（附注释）

# //银行家算法

# #include "stdafx.h"

# #include <iostream>

# using namespace std;

# #define MAXN 100

# int Available[MAXN]; //各资源未分配的资源数

# int Max[MAXN][MAXN]; //每个进程最多所需的资源数

# int Allocation[MAXN][MAXN]; //每个进程已占有的资源数

# int Need[MAXN][MAXN]; //每个进程还需的资源数

# int Request[MAXN]; //进程请求的各资源数

# int Finish[MAXN];

# int path[MAXN] = { 0 }; //安全序列

# int PNum, RNum; //进程数，资源数

# void input(int PNum, int RNum) {

# int i,j;

# cout << "输入每个进程最多所需的资源数：\n";

# for (i = 0; i < PNum; i++) {

# cout << "P" << i << " ：";

# for (j = 0; j < RNum; j++)

# cin >> Max[i][j];

# }

# cout << "输入每个进程已占有的资源数：\n";

# for (i = 0; i < PNum; i++) {

# cout << "P" << i << " ：";

# for (j = 0; j < RNum; j++) {

# cin >> Allocation[i][j];

# //每个进程还需的资源数Need

# Need[i][j] = Max[i][j] - Allocation[i][j];

# if (Need[i][j] < 0) {

# cout << "输入的进程P" << i << "所拥有的第" << j + 1 << "个资源数不能小于0，请重新输入：\n";

# j--;

# continue;

# }

# }

# }

# cout << "每个进程还需的资源数：\n";

# for (i = 0; i < PNum; i++) {

# cout << "P" << i << " ：";

# for (j = 0; j < RNum; j++) {

# cout << " " << Need[i][j];

# }

# cout << endl;

# }

# cout << "输入未分配的资源数：\n";

# for (i = 0; i < RNum; i++)

# cin >> Available[i];

# }

# //判断是否存在安全序列

# int Safe() {

# int i, j, jj, k = 0;

# int Available\_copy[MAXN]; //未分配的资源数

# int Allocation\_copy[MAXN][MAXN]; //已占有的资源数

# int Need\_copy[MAXN][MAXN]; //进程还需的资源数

# 

# for (i = 0; i < RNum; i++)

# Available\_copy[i] = Available[i];

# for (i = 0; i < PNum; i++) {

# for (j = 0; j < RNum; j++)

# Allocation\_copy[i][j] = Allocation[i][j];

# }

# for (i = 0; i < PNum; i++) {

# for (j = 0; j < RNum; j++)

# Need\_copy[i][j] = Need[i][j];

# }

# for (i = 0; i < PNum; i++)

# Finish[i] = 0;

# //遍历每个进程

# for (i = 0; i < PNum; i++) {

# if (Finish[i] == 1)

# continue;

# else {

# //前提：需要的资源<未分配的资源

# for (j = 0; j < RNum; j++) {

# if (Need\_copy[i][j] > Available\_copy[j])

# break;

# }

# //若找到一个进程的所有资源请求满足要求，更新

# if (j == RNum) {

# Finish[i] = 1;

# for (jj = 0; jj < RNum; jj++){

# Available\_copy[jj] += Allocation[i][jj];

# Allocation\_copy[i][jj] = 0;

# Need\_copy[i][jj] = 0;

# }

# int ii;

# cout << "Detail process" << k << ":\n" << "满足P" << i << "请求，更新资源列表";

# cout << "\n进程已分配到的资源数：\n";

# for (ii = 0; ii < PNum; ii++) {

# cout << "P" << ii << " ：";

# for (jj = 0; jj < RNum; jj++)

# cout << " " << Allocation\_copy[ii][jj];

# cout << endl;

# }

# cout << "进程还需的资源数：\n";

# for (ii = 0; ii < PNum; ii++) {

# cout << "P" << ii << " ：";

# for (jj = 0; jj < RNum; jj++)

# cout << " " << Need\_copy[ii][jj];

# cout << endl;

# }

# cout << "未分配的资源数：";

# for (jj = 0; jj < RNum; jj++)

# cout << " " << Available\_copy[jj];

# cout << endl;

# 

# path[k++] = i + 1; //写入安全序列

# i = -1; //NOTE

# }

# else

# continue;

# }

# if (k == PNum) { //存在安全序列

# return 1;

# }

# }

# return 0;

# }

# //输出安全序列

# void outPath() {

# cout << "安全序列：P" << path[0] - 1;

# for (int i = 1; path[i] != 0; i++) {

# cout << "->P" << path[i] - 1;

# }

# //清空path，为下次存入安全序列做准备

# for (int j = 0; j < PNum; j++)

# path[j] = 0;

# cout << endl;

# }

# int requestP() {

# int n,i;

# cout << "输入要申请资源的进程号：";

# cin >> n;

# cout << "输入进程请求的各资源数：";

# for (i = 0; i < RNum; i++)

# cin >> Request[i];

# for (i = 0; i < RNum; i++) {

# if (Request[i] > Need[n][i]) {

# cout << "Error：请求的资源数大于进程所需的资源数！\n";

# return 1;

# }

# if (Request[i] > Available[i]) {

# cout << "请求资源数大于未分配的资源数，系统无足够资源，进程P" << n << "阻塞等待...\n";

# return 1;

# }

# }

# //更新未分配、已分配和还需的资源数

# for (i = 0; i < RNum; i++) {

# Available[i] -= Request[i];

# Allocation[n][i] += Request[i];

# Need[n][i] -= Request[i];

# }

# if (Safe()) {

# cout << "存在安全序列，系统处于安全状态，同意分配请求！\n";

# outPath();

# }

# //若不存在安全序列，还原未分配、已分配和还需的资源数

# else {

# for (i = 0; i < RNum; i++) {

# Available[i] += Request[i];

# Allocation[n][i] -= Request[i];

# Need[n][i] += Request[i];

# }

# cout << "不存在安全序列，系统处于不安全状态，拒绝请求！\n";

# }

# return 0;

# }

# void print() {

# int i, j;

# cout << "\n进程已分配到的资源数：\n";

# for (i = 0; i < PNum; i++) {

# cout << "P" << i << " ：";

# for (j = 0; j < RNum; j++)

# cout << " " << Allocation[i][j];

# cout << endl;

# }

# cout << "进程还需的资源数：\n";

# for (i = 0; i < PNum; i++) {

# cout << "P" << i << " ：";

# for (j = 0; j < RNum; j++)

# cout << " " << Need[i][j];

# cout << endl;

# }

# cout << "未分配的资源数：";

# for (j = 0; j < RNum; j++)

# cout << " " << Available[j];

# cout << endl;

# }

# int main(){

# cout<<"----------银行家算法----------\n";

# cout << "输入进程数：";

# cin >> PNum;

# cout << "输入资源种类：";

# cin >> RNum;

# input(PNum, RNum); //读取输入

# cout << "RESULT：\n";

# if (Safe()) {

# cout << "存在安全序列，系统处于安全状态！\n";

# outPath();

# }

# else {

# cout << "不存在安全序列，系统处于不安全状态！\n";

# return 0;

# }

# cout << endl;

# while (true) {

# requestP();

# print();

# string s;

# cout << "是否再次请求分配？请输入yes或no：\n";

# while (true) {

# cin >> s;

# if (s == "yes" || s == "no")

# break;

# else {

# cout << "请按要求重新输入：\n";

# continue;

# }

# }

# if (s == "yes")

# continue;

# else

# break;

# }

# return 0;

# }

# 五、实验结果

（1）输入已知条件，判断是否存在安全序列使得系统处于安全状态。



图5-1-1 输入资源情况

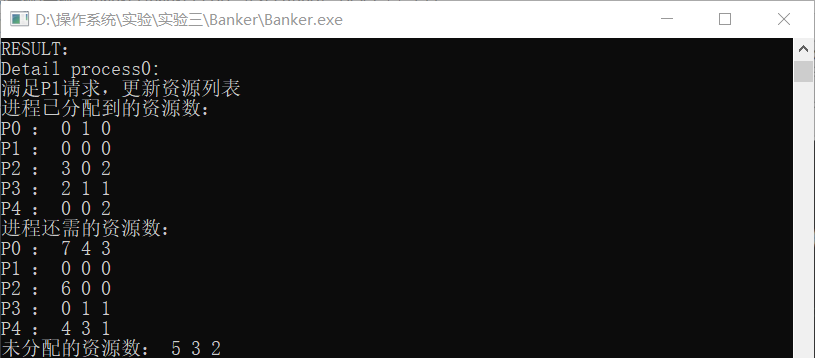


图5-1-2 判断安全序列过程1



图5-1-3 判断安全序列过程2



图5-1-4 判断安全序列过程3



图5-1-5 判断安全序列过程4



图5-1-6 判断安全序列过程5



图5-1-7 系统处于安全状态

（2）P1请求资源Request1(1,0,2)，首先考虑前提条件：

 1）Request1(1,0,2)≤Need1(1,2,2)，P1请求在最大需求范围内。

 2）Request1(1,0,2)≤ Available(3,3,2)，未分配的资源数可满足P1请求。

 若1、2条件都满足，则假设把资源分配给进程P1并修改有关数据：

Available = Available(3，3，2)－Request1(1,0,2) =(2,3,0)；

Need1 = Need1(1,2,2)－Request1(1,0,2)= (0,2,0)；

Allocation1 =Allocation1 (2,0,0)+Request1(1,0,2) =(3,0,2)；

最后检测是否存在安全序列使得系统处于安全状态。

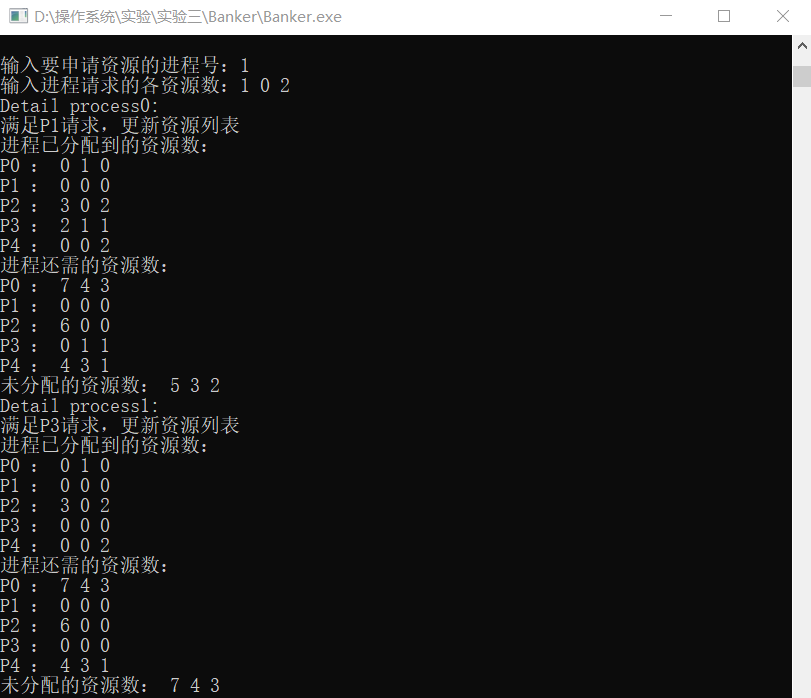


图5-2-1 P1提出请求

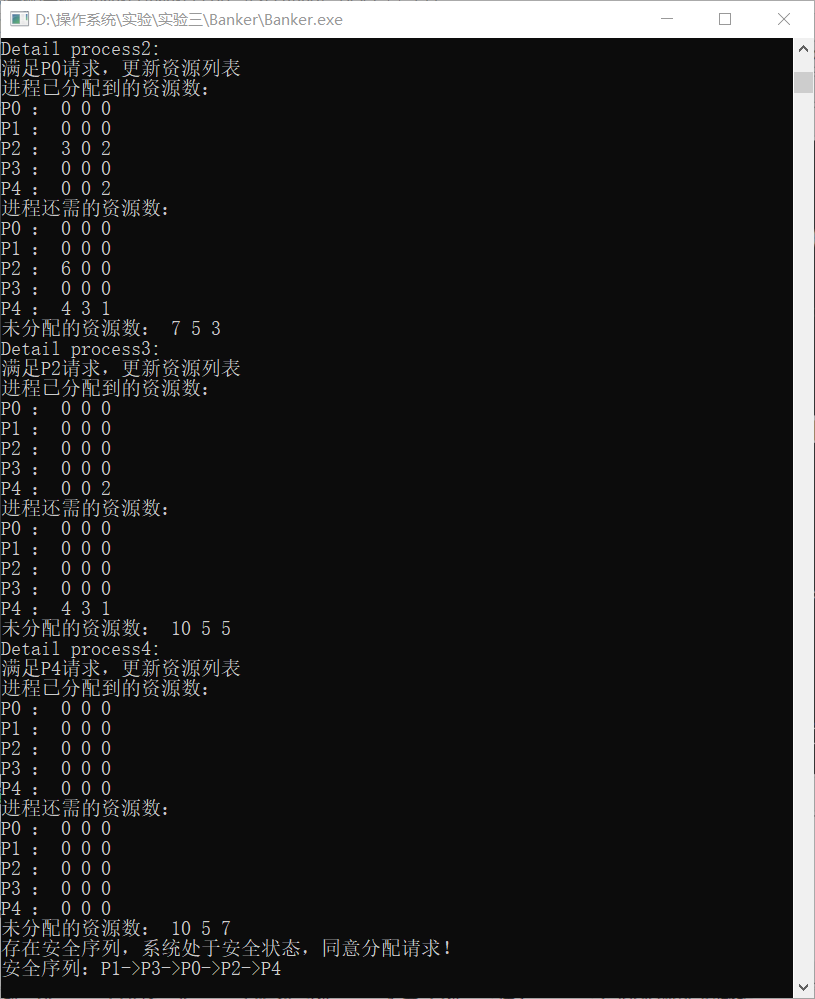


图5-2-2 同意P1提出请求

（3）P4请求资源Request4(3,3,0)，首先考虑前提条件：

 Request4(3,3,0)≤Need4(4,3,1)，P4请求在最大需求范围内。

 Request4(3,3,0)≤Available(2,3,0)不成立，即可用资源不能满足P4请求，进程P4阻塞等待。

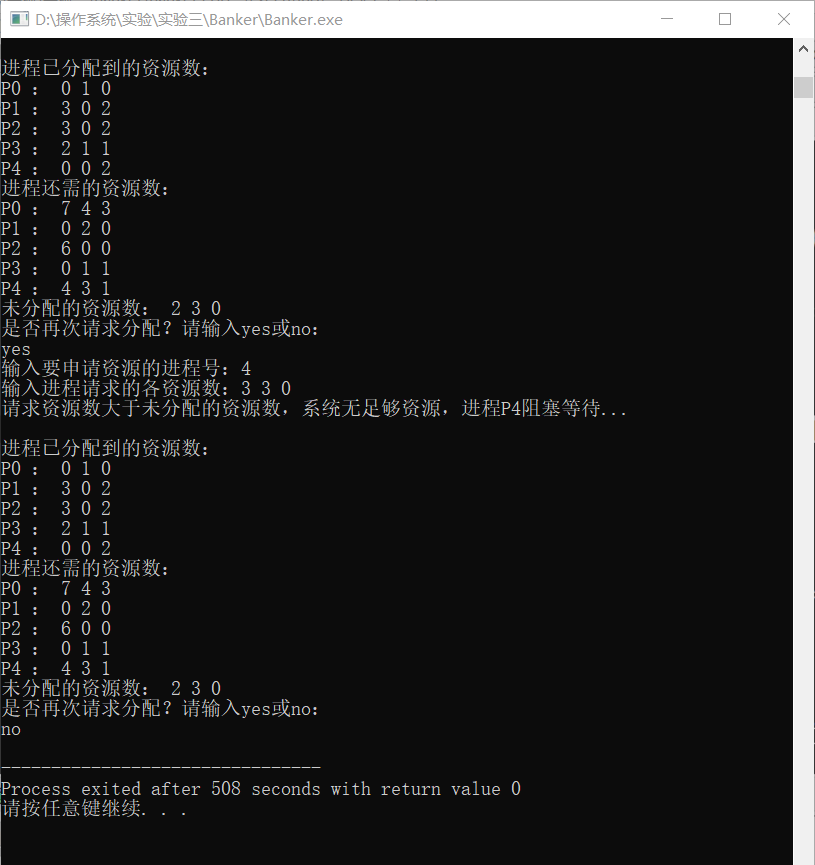


图5-3-1 拒绝P4提出的请求，阻塞

# 六、实验中遇到的问题及解决方法

# 本实验中，我在判断是否存在安全序列的算法Safe()中遇到了一点小问题，具体问题以及解决方案如下。

# 第一个问题是我在Safe中进行判断时直接使用了全局变量Available[MAXN]造成错误，我通过声明一个未分配资源数组的拷贝对象Available\_copy[MAXN]得以解决。同时，为了更加具体地显示判断过程，我在函数中对Allocation\_copy和Need\_copy数组也进行了拷贝，方便打印出每个判断过程中未分配的资源数、进程已占有的资源数、进程还需的资源数的动态变化。

（2）第二个问题是当我找到一个进程的所有资源请求都满足要求时，更新出错，打印出的安全序列不符合预期。经过排查，我发现当我把进程号写入安全序列path[]后，没有对i变量进行更新，导致整个循环只遍历一遍。于是，我写入安全序列后设置i = 0; 但结果还是不理想，经过思考，我发现重新进入for循环时会直接执行i++，导致进程P0没有被遍历，因此，修改i = -1; 后成功。

# 七、实验体会

（1）本实验要求设计并实现银行家算法，银行家算法是一种最有代表性的避免[死锁](https://baike.baidu.com/item/%E6%AD%BB%E9%94%81" \t "https://baike.baidu.com/item/%E9%93%B6%E8%A1%8C%E5%AE%B6%E7%AE%97%E6%B3%95/_blank)的算法，其核心思想是：进程动态地申请资源，每次申请资源之前，系统都执行安全状态检查算法Safe()判断本次请求是否会使系统处于不安全状态，如果不安全则阻塞进程；如果存在一个安全序列，则系统处于安全状态，同意请求，完成资源分配。因此，为实现银行家算法，系统必须设置若干[数据结构](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84" \t "https://baike.baidu.com/item/%E9%93%B6%E8%A1%8C%E5%AE%B6%E7%AE%97%E6%B3%95/_blank)。

（2）安全状态不是死锁状态；死锁状态是不安全状态。不安全状态可能导致死锁，但不安全状态不一定是死锁状态。只要存在一个安全序列，系统就处于安全状态，不会产生死锁。