

### 

### 实验报告

**课程名称:** **实用操作系统教程**

**设计课题: 银行家算法问题**

**指导教师: 熊婷**

**专 业：** 计算机科学与技术

**班 级：** 1982066班

**学 号：** 198206606

**姓 名：** 陈文龙

**成 员：** 陈文龙

**二O 二 一 年 12 月 4 日**

**目 录**

[1.实验目的 1](#_Toc90209674)

[2.实验内容和要求 1](#_Toc90209675)

[3.算法描述 1](#_Toc90209676)

[4.算法流程图 3](#_Toc90209677)

[5.源程序及注释 3](#_Toc90209678)

[**5.3源程序 4**](#_Toc90209680)

[**5.4结果 8**](#_Toc90209681)

[6.总结 8](#_Toc90209682)

[参考文献： 8](#_Toc90209683)

# 1.实验目的

银行家算法是避免死锁的一种重要方法，通过编写一个简单的银行家算法程序，加深了解有关资源申请、避免死锁等概念，并体会和了解死锁和避免死锁的具体实施方法。

# 2.实验内容和要求

自选语言编写和调试一个系统动态资源的简单模拟程序，观察死锁产生的条件，并使用适当的算法，有效的防止和避免死锁的发生。

要求如下：

（1） 模拟一个银行家算法；

（2） 初始化时让系统拥有一定的资源；

（3） 用键盘输入的方式申请资源；

（4） 如果预分配后，系统处于安全状态，则修改系统的资源分配情况；

（5） 如果预分配后，系统处于不安全状态，则提示不能满足请求。

# 3.算法描述

1、算法思路：先对用户提出的请求进行合法性检查，即检查请求是否大于需要的，是否大于可利用的。若请求合法，则进行预分配，对分配后的状态调用安全性算法进行检查。若安全，则分配；若不安全，则拒绝申请，恢复到原来的状态，拒绝申请。

2、银行家算法步骤：

设进程i提出请求Request[n]，则银行家算法按如下规则进行判断。

(1)如果Request[n]>Need[i，n]，则报错返回。

(2)如果Request[n]>Available，则进程i进入等待资源状态，返回。

(3)假设进程i的申请已获批准，于是修改系统状态：

Available=Available-Request

Allocation=Allocation+Request

Need=Need-Request

(4)系统执行安全性检查，如安全，则分配成立；否则试探险性分配作废，系统恢复原状，进程等待。

3、安全性算法步骤：

（1）设置两个向量

①工作向量Work。它表示系统可提供进程继续运行所需要的各类资源数目，执行安全算法开始时，Work=Allocation;

②布尔向量Finish。它表示系统是否有足够的资源分配给进程，使之运行完成，开始时先做Finish[i]=false，当有足够资源分配给进程时，令Finish[i]=true。

（2）从进程集合中找到一个能满足下述条件的进程：

①Finish[i]=false

②Need<=Work

如找到，执行步骤（3）；否则，执行步骤（4）。

（3）当进程P获得资源后，可顺利执行，直至完成，并释放出分配给它的资源，故应执行：

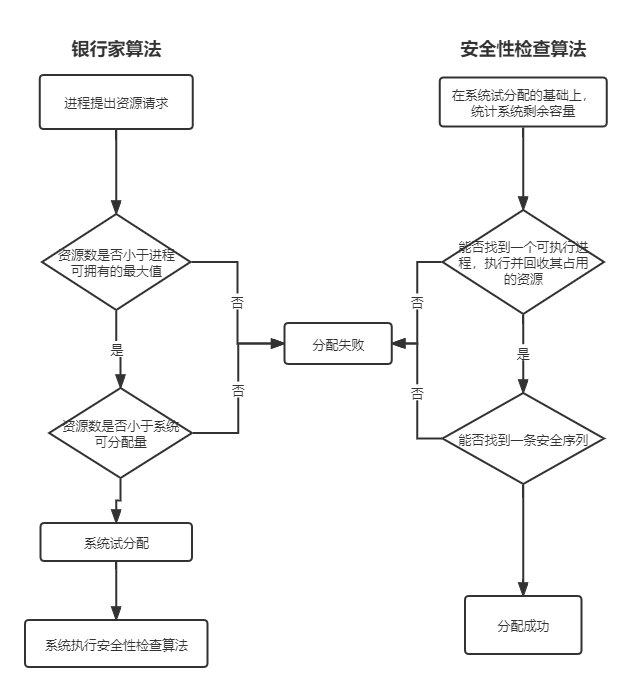
Work=Work+Allocation;

Finish[i]=true;

转向步骤（2）。

（4）如果所有进程的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

# 4.算法流程图

****

# 5.源程序及注释

程序中使用的数据结构及符号说明。  
**5.1全局变量定义**  
int Available[100]; //可利用资源向量Available ,是个含有m个元素的数组，其中的每一个元素代表一类可利用的资源数目。如果Available[j] = K，则表示系统中现有Rj类资源K个。  
int Max[50][100];   //最大需求矩阵Max 这是一个n×m的矩阵，它定义了系统中n个进程中的每一个进程对m类资源的最大需求。如果Max[i, j] = K，则表示进程i需要Rj类资源的最大数目为K。  
int Allocation[50][100];  //已经分配矩阵  
int Need[50][100];        //进程需求矩阵  
int Request[50][100];     //M个进程还需要N类资源的资源量  
int Finish[50];//  
int p[50];  
int m, n;   //M个进程,N类资源  
**5.2程序包含函数:**

//安全性算法  
int Safe();

**5.3源程序**

#include <iostream>

using namespace std;

//全局变量定义

int Available[100]; //可利用资源向量Available ,是个含有m个元素的数组，其中的每一个元素代表一类可利用的资源数目。如果Available[j] = K，则表示系统中现有Rj类资源K个。

int Max[50][100]; //最大需求矩阵Max 　　这是一个n×m的矩阵，它定义了系统中n个进程中的每一个进程对m类资源的最大需求。如果Max[i, j] = K，则表示进程i需要Rj类资源的最大数目为K。

int Allocation[50][100]; //已经分配矩阵

int Need[50][100]; //进程需求矩阵

int Request[50][100]; //M个进程还需要N类资源的资源量

int Finish[50];//

int p[50];

int m, n; //M个进程,N类资源

//安全性算法3

int Safe()

{

int i, j, l = 0;

int Work[100]; //可提供给进程各类资源资源数组

for (i = 0; i < n; i++)

Work[i] = Available[i];//在执行安全算法开始时，可提供的各类资源数目=系统现有各类资源数目;Work=Available;

for (i = 0; i < m; i++)

Finish[i] = 0;//表示系统是否有足够的资源分配给进程

for (i = 0; i < m; i++)

{

if (Finish[i] == 1)//工作向量等于1,进程执行

continue;

else

{

for (j = 0; j < n; j++)

{

if (Need[i][j] > Work[j])

break;

}

if (j == n)

{

Finish[i] = 1;

for (int k = 0; k < n; k++)

Work[k] = Work[k] + Allocation[i][k];//直至完成，并释放出分配给它的资源，故应执行可提供资源数量更新

p[l++] = i;

i = -1;

}

else continue;

}

if (l == m)

{

cout << "系统是安全的" << '\n';

cout << "系统安全序列是:\n";

for (i = 0; i < l; i++)

{

cout << p[i];

if (i != l - 1)

cout << "-->";

}

cout << '\n';

return 1;

}

}

return 0;

}

//银行家算法

int main()

{

int i, j, mi;

cout << "输入进程的数目:\n";

cin >> m;

cout << "输入资源的种类:\n";

cin >> n;

cout << "输入每个进程最多所需的各类资源数,按照" << m << "x" << n << "矩阵输入\n";

for (i = 0; i < m; i++)

for (j = 0; j < n; j++)

cin >> Max[i][j];//将用户输入的资源需求放进最大需求矩阵数组里

cout << "输入每个进程已经分配的各类资源数,按照" << m << "x" << n << "矩阵输入\n";

for (i = 0; i < m; i++)

{

for (j = 0; j < n; j++)

{

cin >> Allocation[i][j];//分配矩阵

Need[i][j] = Max[i][j] - Allocation[i][j];//进程需要的各类资源数=进程最大需求资源数-已分配资源数

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

{

cout << "你输入的第" << i + 1 << "个进程所拥有的第" << j + 1 << "个资源错误，请重新输入:\n";

j--;

continue;

}

}

}

cout << "请输入各个资源现有的数目:\n";

for (i = 0; i < n; i++)

cin >> Available[i];

Safe();

while (1)

{

cout << "输入要申请的资源的进程号：（第一个进程号为0，第二个进程号为1，依此类推）\n";

cin >> mi;

cout << "输入进程所请求的各个资源的数量\n";

for (i = 0; i < n; i++)

cin >> Request[mi][i];

for (i = 0; i < n; i++)

{

if (Request[mi][i] > Need[mi][i])

{

cout << "所请求资源数超过进程的需求量！\n";

return 0;

}

if (Request[mi][i] > Available[i])

{

cout << "所请求资源数超过系统所有的资源数！\n";

return 0;

}

}

for (i = 0; i < n; i++)

{

Available[i] = Available[i] - Request[mi][i];//可利用资源=可利用资源-进程需要

Allocation[mi][i] = Allocation[mi][i] + Request[mi][i];//已分配资源=已分配资源+进程需要

Need[mi][i] = Need[mi][i] - Request[mi][i];//进程还需资源=进程还需资源-进程需要

}

if (Safe())

cout << "同意分配请求\n";

else

{

cout << "对不起.你的请求被拒绝…\n";

for (i = 0; i < n; i++)

{

Available[i] = Available[i] - Request[mi][i];//可利用资源=可利用资源-进程需要

Allocation[mi][i] = Allocation[mi][i] + Request[mi][i];//已分配资源=已分配资源+进程需要

Need[mi][i] = Need[mi][i] - Request[mi][i];//进程还需资源=进程还需资源-进程需要

}

}

for (i = 0; i < m; i++)

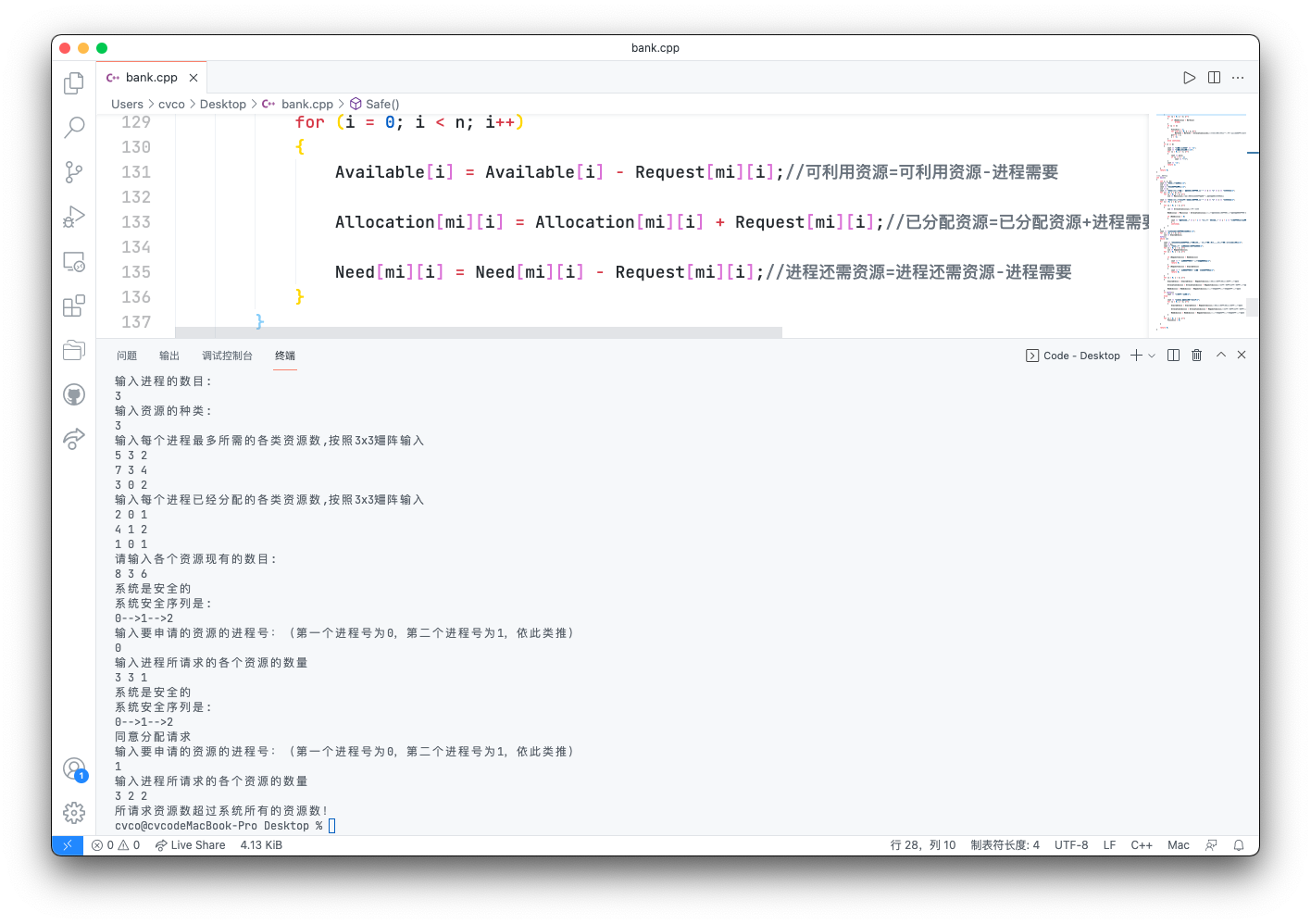
Finish[i] = 0;

}

return 0;

}

**5.4结果**

****

# 6.总结

银行家算法是一种用来避免操作系统死锁出现的有效算法。死锁：是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。安全状态一定是没有死锁发生。不安全状态：不存在一个安全序列。不安全状态不一定导致死锁。

参考文献：

［1］李建伟主编. 实用操作系统教程[M]. 北京：清华大学出版社，2016