**操作系统实验**

**实验报告**

题目：[实验三 预防进程死锁的银行家算法]

**专业：[软件工程]**

**年级：[2017级]**

**姓名：[陈思翔]**

**学号：[1725121003]**

目录

[一、需求分析 4](#_Toc27926703)

[1、程序设计的任务和目的 4](#_Toc27926704)

[2、输入的形式和输入值的范围 4](#_Toc27926705)

[3、输出的形式 4](#_Toc27926706)

[4、程序所能达到的功能 4](#_Toc27926707)

[5、测试数据，包括正确的输入及其输出结果和含有错误的输入及其输出结果 4](#_Toc27926708)

[正确用例 4](#_Toc27926709)

[错误用例（输入负值，影响正确结果） 6](#_Toc27926710)

[二、概要设计 6](#_Toc27926711)

[1、抽象数据类型的定义 6](#_Toc27926712)

[2、主程序的流程 7](#_Toc27926713)

[3、各程序模块之间的层次(调用)关系 7](#_Toc27926714)

[三、详细设计 9](#_Toc27926715)

[1、Order银行家算法排序 9](#_Toc27926716)

[2、NewRequest发起新请求 10](#_Toc27926717)

[3、NewFinish函数判断进程完成 10](#_Toc27926718)

[4、IsExecutable函数判断资源充足与否 10](#_Toc27926719)

[5、IsSafe函数判断当前进程是否安全 11](#_Toc27926720)

[四、调试分析 11](#_Toc27926721)

[调试过程中遇到的问题以及解决方法，设计与实现的回顾讨论和分析 11](#_Toc27926722)

[算法的性能分析(包括基本操作和其它算法的时间复杂度和空间复杂度的分析)及其改进设想 11](#_Toc27926723)

[性能分析 11](#_Toc27926724)

[改进设想 11](#_Toc27926725)

[五、用户使用说明 11](#_Toc27926726)

[使用说明 11](#_Toc27926727)

[六、测试结果 12](#_Toc27926728)

[测试结果，包括输入和输出 12](#_Toc27926729)

[七、附录 14](#_Toc27926730)

[带注释的源程序 14](#_Toc27926731)

# 一、需求分析

## 1、程序设计的任务和目的

通过这次实验，加深对进程死锁的理解，进一步掌握进程资源的分配、死锁的检测和安全序列的生成方法。

## 2、输入的形式和输入值的范围

进程个数n，资源种类m，T0时刻各个进程的资源分配情况（可以运行输入，也可以在程序中设置）

## 3、输出的形式

如果安全，输出安全的进程序列，不安全则提示信息。

## 4、程序所能达到的功能

程序模拟预防进程死锁的银行家算法的工作过程。假设系统中有n个进程P1, … ,Pn，有m类可分配的资源R1, … ,Rm，在T0时刻，进程Pi分配到的j类资源为Allocationij个，它还需要j类资源Need ij个，系统目前剩余j类资源Workj个，现采用银行家算法进行进程资源分配预防死锁的发生。

## 5、测试数据，包括正确的输入及其输出结果和含有错误的输入及其输出结果

### 正确用例

#### 输入

请输入进程个数n：5  
请输入资源种类m：3  
请输入进程1的资源1的最大允许数量Max[1]：7  
请输入进程1的资源2的最大允许数量Max[2]：5  
请输入进程1的资源3的最大允许数量Max[3]：3  
请输入进程1的资源1的已分配数量Allocation[1]：0  
请输入进程1的资源2的已分配数量Allocation[2]：1  
请输入进程1的资源3的已分配数量Allocation[3]：0  
请输入进程2的资源1的最大允许数量Max[1]：3  
请输入进程2的资源2的最大允许数量Max[2]：2  
请输入进程2的资源3的最大允许数量Max[3]：2  
请输入进程2的资源1的已分配数量Allocation[1]：2  
请输入进程2的资源2的已分配数量Allocation[2]：0  
请输入进程2的资源3的已分配数量Allocation[3]：0  
请输入进程3的资源1的最大允许数量Max[1]：9  
请输入进程3的资源2的最大允许数量Max[2]：0  
请输入进程3的资源3的最大允许数量Max[3]：2  
请输入进程3的资源1的已分配数量Allocation[1]：3  
请输入进程3的资源2的已分配数量Allocation[2]：0  
请输入进程3的资源3的已分配数量Allocation[3]：2  
请输入进程4的资源1的最大允许数量Max[1]：2  
请输入进程4的资源2的最大允许数量Max[2]：2  
请输入进程4的资源3的最大允许数量Max[3]：2  
请输入进程4的资源1的已分配数量Allocation[1]：2  
请输入进程4的资源2的已分配数量Allocation[2]：1  
请输入进程4的资源3的已分配数量Allocation[3]：1  
请输入进程5的资源1的最大允许数量Max[1]：4  
请输入进程5的资源2的最大允许数量Max[2]：3  
请输入进程5的资源3的最大允许数量Max[3]：3  
请输入进程5的资源1的已分配数量Allocation[1]：0  
请输入进程5的资源2的已分配数量Allocation[2]：0  
请输入进程5的资源3的已分配数量Allocation[3]：2  
请输入可用资源1的可用数量：3  
请输入可用资源2的可用数量：3  
请输入可用资源3的可用数量：2

#### 输出

SafeOrder为： 进程 2 进程 4 进程 5 进程 1 进程 3  
  
剩余资源情况为：  
资源 1剩余数量： 3  
资源 2剩余数量： 3  
资源 3剩余数量： 2  
  
是否还有新请求,若有请输入进程序号，若无请输入0：2  
请输入进程2需求的资源1的数量:1  
请输入进程2需求的资源2的数量:0  
请输入进程2需求的资源3的数量:2  
SafeOrder为： 进程 2 进程 4 进程 5 进程 1 进程 3  
  
剩余资源情况为：  
资源 1剩余数量： 2  
资源 2剩余数量： 3  
资源 3剩余数量： 0

### 错误用例（输入负值，影响正确结果）

#### 输入

请输入进程个数n：1  
请输入资源种类m：3  
请输入进程1的资源1的最大允许数量Max[1]：1  
请输入进程1的资源2的最大允许数量Max[2]：1  
请输入进程1的资源3的最大允许数量Max[3]：1  
请输入进程1的资源1的已分配数量Allocation[1]：-1  
请输入进程1的资源2的已分配数量Allocation[2]：-1  
请输入进程1的资源3的已分配数量Allocation[3]：-1  
请输入可用资源1的可用数量：2  
请输入可用资源2的可用数量：2  
请输入可用资源3的可用数量：2

#### 输出

SafeOrder为： 进程 1  
  
剩余资源情况为：  
资源 1剩余数量： 2  
资源 2剩余数量： 2  
资源 3剩余数量： 2  
  
是否还有新请求,若有请输入进程序号，若无请输入0：1  
请输入进程1需求的资源1的数量:2  
请输入进程1需求的资源2的数量:2  
请输入进程1需求的资源3的数量:2  
SafeOrder为： 进程 1  
  
剩余资源情况为：  
资源 1剩余数量： 0  
资源 2剩余数量： 0  
资源 3剩余数量： 0

# 二、概要设计

## 1、抽象数据类型的定义

int n;//进程个数n  
int m;//资源种类m  
int Available[MaxNumber];//可用资源数量  
int Request[MaxNumber];//进程请求的资源数量  
int SafeOrder[MaxNumber];//安全进程序列  
  
//定义进程的数据结构  
typedef struct {  
 int Max[MaxNumber];  
 int Allocation[MaxNumber];  
 int Need[MaxNumber];  
 bool Finished;//完成状态  
} Progress;

## 2、主程序的流程

int main() {  
 BankerAlgorithm bankerAlgorithm{};  
 bankerAlgorithm.Input();  
 bankerAlgorithm.Order(bankerAlgorithm.progress, bankerAlgorithm.Available);  
  
 return 0;  
}

## 3、各程序模块之间的层次(调用)关系

int main() {  
 BankerAlgorithm bankerAlgorithm{};  
 bankerAlgorithm.Input();  
 bankerAlgorithm.Order(bankerAlgorithm.progress, bankerAlgorithm.Available);  
  
 return 0;  
}  
  
//输入函数调用InputAlgorithm函数选择输入函数  
void Input() {  
 ···  
}  
  
//进行安全进程检测并输出安全序列  
void Order(Progress pg[MaxNumber], int a[MaxNumber]) {  
 //定义下标数  
 int orderNum = 0;  
  
 //复制进程操作  
 Progress progressCopy[MaxNumber];  
 for (int i = 1; i <= n; i++) {  
 progressCopy[i] = pg[i];  
 }  
  
 //复制可用需求数  
 int available[MaxNumber];  
 for (int i = 1; i <= m; i++) {  
 available[i] = a[i];  
 }  
  
 //调用NewFinish函数判断所有进程是否完成  
 while (!NewFinish(progressCopy)) {  
 //调用IsSafe函数判断现在是否安全  
 if (IsSafe(progressCopy, available)) {  
 for (int i = 1; i <= n; i++) {  
 if (!progressCopy[i].Finished &&  
 //调用IsExecutable函数判断可分配资源可否满足该进程  
 IsExecutable(progressCopy[i], available)) {  
 //只有同时满足进程未完成、可分配需求足够才进行分配  
 ···  
 }  
 }  
 } else {  
 cout << "不安全" << endl;  
 exit(0);  
 }  
 }  
  
 //输出SafeOrder  
 ···  
  
 NewRequest(pg, a);  
}  
  
//判断是否有新请求，若无则退出，若有则输入其请求资源数  
void NewRequest(Progress pg[MaxNumber], int a[MaxNumber]) {  
 ···  
   
 //输出剩余资源数量参考  
 ···  
  
 //判断新请求，若有可用新请求，调用Order函数重新进行银行家算法安全计算  
 cout << endl << "是否还有新请求,若有请输入进程序号，若无请输入0：";  
 cin >> newRequest;  
 if (newRequest != 0) {  
 for (int i = 1; i <= m; i++) {  
 //若可分配资源不足或超出需求，则重新调用NewRequest函数  
 NewRequest(pg, a);  
 }  
 ···  
 Order(pg, a);  
 } else {  
 return;  
 }  
}  
  
//判断所有进程是否完成  
bool NewFinish(Progress pg[MaxNumber]) {  
 ···  
}  
  
//判断可分配资源可否满足该进程  
bool IsExecutable(Progress pg, const int a[MaxNumber]) {  
 ···  
}  
  
//判断现在是否安全  
bool IsSafe(Progress pg[MaxNumber], int a[MaxNumber]) {  
 ···  
}

# 三、详细设计

**实现程序模块的具体算法**

## 1、Order银行家算法排序

//进行安全进程检测并输出安全序列  
void Order(Progress pg[MaxNumber], int a[MaxNumber]) {  
 //定义下标数  
 int orderNum = 0;  
  
 //复制进程操作  
 ···  
  
 //复制可用需求数  
 ···  
  
 //调用NewFinish函数判断所有进程是否完成  
 while (!NewFinish(progressCopy)) {  
 //调用IsSafe函数判断现在是否安全  
 if (IsSafe(progressCopy, available)) {  
 for (int i = 1; i <= n; i++) {  
 if (!progressCopy[i].Finished &&  
 //调用IsExecutable函数判断可分配资源可否满足该进程  
 IsExecutable(progressCopy[i], available)) {  
 //只有同时满足进程未完成、可分配需求足够才进行分配  
 ···  
 }  
 }  
 } else {  
 cout << "不安全" << endl;  
 exit(0);  
 }  
 }  
  
 //输出SafeOrder  
 ···  
 //若为安全状态，重新调用NewRequest函数请求下一个需求输入  
 NewRequest(pg, a);  
}

## 2、NewRequest发起新请求

//判断是否有新请求，若无则退出，若有则输入其请求资源数  
void NewRequest(Progress pg[MaxNumber], int a[MaxNumber]) {  
 ···  
   
 //输出剩余资源数量参考  
 ···  
  
 //判断新请求，若有可用新请求，调用Order函数重新进行银行家算法安全计算  
 cout << endl << "是否还有新请求,若有请输入进程序号，若无请输入0：";  
 cin >> newRequest;  
 if (newRequest != 0) {  
 for (int i = 1; i <= m; i++) {  
 //若可分配资源不足或超出需求，则重新调用NewRequest函数  
 NewRequest(pg, a);  
 }  
 ···  
 Order(pg, a);  
 } else {  
 return;  
 }  
}

## 3、NewFinish函数判断进程完成

//判断所有进程是否完成  
bool NewFinish(Progress pg[MaxNumber]) {  
 bool finished = true;  
 for (int i = 1; i <= n; i++) {  
 finished \*= pg[i].Finished;  
 }  
 return finished;  
}

## 4、IsExecutable函数判断资源充足与否

//判断可分配资源可否满足该进程  
bool IsExecutable(Progress pg, const int a[MaxNumber]) {  
 bool isExecutable = true;  
 for (int i = 1; i <= m; i++) {  
 isExecutable \*= (pg.Need[i] <= a[i]);  
 }  
 return isExecutable;  
}

## 5、IsSafe函数判断当前进程是否安全

//判断现在是否安全  
bool IsSafe(Progress pg[MaxNumber], int a[MaxNumber]) {  
 bool isExecutable = false;  
 for (int i = 1; i <= n; i++) {  
 if (IsExecutable(pg[i], a)) {  
 isExecutable = true;  
 return isExecutable;  
 }  
 }  
 return isExecutable;  
}

# 四、调试分析

## 调试过程中遇到的问题以及解决方法，设计与实现的回顾讨论和分析

## 算法的性能分析(包括基本操作和其它算法的时间复杂度和空间复杂度的分析)及其改进设想

### 性能分析

|  |  |  |
| --- | --- | --- |
| 算法 | 时间复杂度 | 空间复杂度 |
| Order算法安全进程检测 | T(n) = O(n2) | S(n) = O(n) |
| NewRequest算法请求资源 | T(n) = O(n2) | S(n) = O(n2) |
| NewFinish算法判断所有进程是否完成 | T(n) = O(n) | S(n) = O(n) |
| IsExecutable算法判断可分配资源可否满足该进程 | T(n) = O(n) | S(n) = O(n) |
| IsSafe算法判断进程是否处于安全状态 | T(n) = O(n) | S(n) = O(n) |

### 改进设想

输出较为繁复，多次测试较为不便，可改为读取文件输入

# 五、用户使用说明

## 使用说明

* 控制台会提示要求用户进行输入，按提示输入内容即可
* 不要输入负值，将影响正确结果
* 选择算法输入完成后将会输出安全的进程序列
* 可根据需要选择是否使用进行其他资源请求或退出

# 六、测试结果

## 测试结果，包括输入和输出

D:\Documents\MyCourse\OperatingSystem\cmake-build-debug\chapter03.exe  
请输入进程个数n：5  
请输入资源种类m：3  
请输入进程1的资源1的最大允许数量Max[1]：7  
请输入进程1的资源2的最大允许数量Max[2]：5  
请输入进程1的资源3的最大允许数量Max[3]：3  
请输入进程1的资源1的已分配数量Allocation[1]：0  
请输入进程1的资源2的已分配数量Allocation[2]：1  
请输入进程1的资源3的已分配数量Allocation[3]：0  
请输入进程2的资源1的最大允许数量Max[1]：3  
请输入进程2的资源2的最大允许数量Max[2]：2  
请输入进程2的资源3的最大允许数量Max[3]：2  
请输入进程2的资源1的已分配数量Allocation[1]：2  
请输入进程2的资源2的已分配数量Allocation[2]：0  
请输入进程2的资源3的已分配数量Allocation[3]：0  
请输入进程3的资源1的最大允许数量Max[1]：9  
请输入进程3的资源2的最大允许数量Max[2]：0  
请输入进程3的资源3的最大允许数量Max[3]：2  
请输入进程3的资源1的已分配数量Allocation[1]：3  
请输入进程3的资源2的已分配数量Allocation[2]：0  
请输入进程3的资源3的已分配数量Allocation[3]：2  
请输入进程4的资源1的最大允许数量Max[1]：2  
请输入进程4的资源2的最大允许数量Max[2]：2  
请输入进程4的资源3的最大允许数量Max[3]：2  
请输入进程4的资源1的已分配数量Allocation[1]：2  
请输入进程4的资源2的已分配数量Allocation[2]：1  
请输入进程4的资源3的已分配数量Allocation[3]：1  
请输入进程5的资源1的最大允许数量Max[1]：4  
请输入进程5的资源2的最大允许数量Max[2]：3  
请输入进程5的资源3的最大允许数量Max[3]：3  
请输入进程5的资源1的已分配数量Allocation[1]：0  
请输入进程5的资源2的已分配数量Allocation[2]：0  
请输入进程5的资源3的已分配数量Allocation[3]：2  
请输入可用资源1的可用数量：3  
请输入可用资源2的可用数量：3  
请输入可用资源3的可用数量：2  
  
SafeOrder为： 进程 2 进程 4 进程 5 进程 1 进程 3  
  
剩余资源情况为：  
资源 1剩余数量： 3  
资源 2剩余数量： 3  
资源 3剩余数量： 2  
  
是否还有新请求,若有请输入进程序号，若无请输入0：2  
请输入进程2需求的资源1的数量:1  
请输入进程2需求的资源2的数量:0  
请输入进程2需求的资源3的数量:2  
SafeOrder为： 进程 2 进程 4 进程 5 进程 1 进程 3  
  
剩余资源情况为：  
资源 1剩余数量： 2  
资源 2剩余数量： 3  
资源 3剩余数量： 0  
  
是否还有新请求,若有请输入进程序号，若无请输入0：4  
请输入进程4需求的资源1的数量:1  
超出需求，请重新输入  
  
  
剩余资源情况为：  
资源 1剩余数量： 2  
资源 2剩余数量： 3  
资源 3剩余数量： 0  
  
是否还有新请求,若有请输入进程序号，若无请输入0：5  
请输入进程5需求的资源1的数量:1  
请输入进程5需求的资源2的数量:3  
请输入进程5需求的资源3的数量:0  
不安全  
  
Process finished with exit code 0

# 七、附录

## 带注释的源程序

#include <iostream>   
#include <iomanip>   
   
using namespace std;   
#define MaxNumber 100   
   
class BankerAlgorithm {   
public:   
   
 int n;//进程个数n   
 int m;//资源种类m   
 int Available[MaxNumber];//可用资源数量   
 int Request[MaxNumber];//进程请求的资源数量   
 int SafeOrder[MaxNumber];//安全进程序列   
   
 //定义进程的数据结构   
 typedef struct {   
 int Max[MaxNumber];   
 int Allocation[MaxNumber];   
 int Need[MaxNumber];   
 bool Finished;//完成状态   
 } Progress;   
   
 Progress progress[MaxNumber];   
   
 //输入进程数、资源种类、各进程有关资源的Max、Allocation、Need数量   
 void Input() {   
 cout << "请输入进程个数n：";   
 cin >> n;   
 cout << "请输入资源种类m：";   
 cin >> m;   
 for (int i = 1; i <= n; i++) {   
 for (int j = 1; j <= m; j++) {   
 cout << "请输入进程" << i << "的资源" << j << "的最大允许数量Max[" << j << "]：";   
 cin >> progress[i].Max[j];   
 }   
 for (int j = 1; j <= m; j++) {   
 cout << "请输入进程" << i << "的资源" << j << "的已分配数量Allocation[" << j << "]：";   
 cin >> progress[i].Allocation[j];   
 }   
 for (int j = 1; j <= m; j++) {   
 progress[i].Need[j] = progress[i].Max[j] - progress[i].Allocation[j];   
 }   
 }   
   
 for (int i = 1; i <= m; i++) {   
 cout << "请输入可用资源" << i << "的可用数量：";   
 cin >> Available[i];   
 }   
   
 cout << endl;   
 }   
   
 //判断是否有新请求，若无则退出，若有则输入其请求资源数   
 void NewRequest(Progress pg[MaxNumber], int a[MaxNumber]) {   
 int newRequest;   
   
 //剩余资源数量参考   
 cout << endl << endl << "剩余资源情况为：" << endl;   
 for (int i = 1; i <= m; i++) {   
 cout << setw(6) << "资源" << setw(2) << i << setw(6) << "剩余数量：" << setw(2) << a[i] << endl;   
 }   
   
 //判断新请求   
 cout << endl << "是否还有新请求,若有请输入进程序号，若无请输入0：";   
 cin >> newRequest;   
 if (newRequest != 0) {   
 for (int i = 1; i <= m; i++) {   
 cout << "请输入进程" << newRequest << "需求的资源" << i << "的数量:";   
 cin >> Request[i];   
 if (Request[i] > a[i]) {   
 cout << "可分配资源不足，请重新输入" << endl;   
 NewRequest(pg, a);   
 }   
 if (Request[i] > pg[newRequest].Need[i]) {   
 cout << "超出需求，请重新输入" << endl;   
 NewRequest(pg, a);   
 }   
 }   
   
 for (int i = 1; i <= m; i++) {   
 a[i] -= Request[i];   
 pg[newRequest].Allocation[i] += Request[i];   
 pg[newRequest].Need[i] -= Request[i];   
 }   
 Order(pg, a);   
 } else {   
 return;   
 }   
 }   
   
 void Order(Progress pg[MaxNumber], int a[MaxNumber]) {   
 //定义下标数   
 int orderNum = 0;   
   
 //复制进程操作   
 Progress progressCopy[MaxNumber];   
 for (int i = 1; i <= n; i++) {   
 progressCopy[i] = pg[i];   
 }   
   
 //复制可用需求数   
 int available[MaxNumber];   
 for (int i = 1; i <= m; i++) {   
 available[i] = a[i];   
 }   
   
 while (!NewFinish(progressCopy)) {//若有   
 if (IsSafe(progressCopy, available)) {   
 for (int i = 1; i <= n; i++) {   
 if (!progressCopy[i].Finished &&   
 IsExecutable(progressCopy[i], available)) {//只有同时满足进程未完成、可分配需求足够才进行分配   
 progressCopy[i].Finished = true;   
 for (int j = 1; j <= m; j++) {   
 available[j] += progressCopy[i].Allocation[j];   
 }   
 SafeOrder[++orderNum] = i;   
 }   
 }   
 } else {   
 cout << "不安全" << endl;   
 exit(0);   
 }   
 }   
   
 //输出SafeOrder   
 cout << "SafeOrder为：";   
 for (int i = 1; i <= n; i++) {   
 cout << setw(12) << "进程" << setw(4) << SafeOrder[i];   
 }   
   
 NewRequest(pg, a);   
 }   
   
 //判断所有进程是否完成   
 bool NewFinish(Progress pg[MaxNumber]) {   
 bool finished = true;   
 for (int i = 1; i <= n; i++) {   
 finished \*= pg[i].Finished;   
 }   
 return finished;   
 }   
   
 //判断可分配资源可否满足该进程   
 bool IsExecutable(Progress pg, const int a[MaxNumber]) {   
 bool isExecutable = true;   
 for (int i = 1; i <= m; i++) {   
 isExecutable \*= (pg.Need[i] <= a[i]);   
 }   
 return isExecutable;   
 }   
   
 //判断现在是否安全   
 bool IsSafe(Progress pg[MaxNumber], int a[MaxNumber]) {   
 bool isExecutable = false;   
 for (int i = 1; i <= n; i++) {   
 if (IsExecutable(pg[i], a)) {   
 isExecutable = true;   
 return isExecutable;   
 }   
 }   
 return isExecutable;   
 }   
};   
   
int main() {   
 BankerAlgorithm bankerAlgorithm{};   
 bankerAlgorithm.Input();   
 bankerAlgorithm.Order(bankerAlgorithm.progress, bankerAlgorithm.Available);   
   
 return 0;   
}