中南大学

《操作系统》

课程实验报告

实验名称 银行家算法模拟

学生姓名 　 白明强

学 号 　 1201170606

专业班级 物联网1802

指导教师 胡小龙

学 院 计算机学院

目录

[实验目的 2](#_Toc40352080)

[实验平台 2](#_Toc40352081)

[详细设计 2](#_Toc40352082)

[结果演示 6](#_Toc40352083)

[正常申请： 6](#_Toc40352084)

[超额申请： 7](#_Toc40352085)

[不安全被拒： 7](#_Toc40352086)

[释放资源： 7](#_Toc40352087)

[完整代码 8](#_Toc40352088)

# 实验目的

死锁是操作系统的设计过程中必须避免的问题。为了了解系统的资源分配情况，假定系统的任何一种资源在任一时刻只能被一个进程使用。任何进程已经占用的资源只能由进程自己释放，而不能由其他进程抢占。当进程申请的资源不能满足时，必须等待。因此，只要资源分配算法能保证进程的资源请求，且不出现循环等待，则系统不会出现死锁。

银行家算法是操作系统中避免死锁的典型算法，本实验可以加深对银行家算法的步骤和相关数据结构用法的更好理解，掌握进程安全性检查的方法与资源分配的方法。

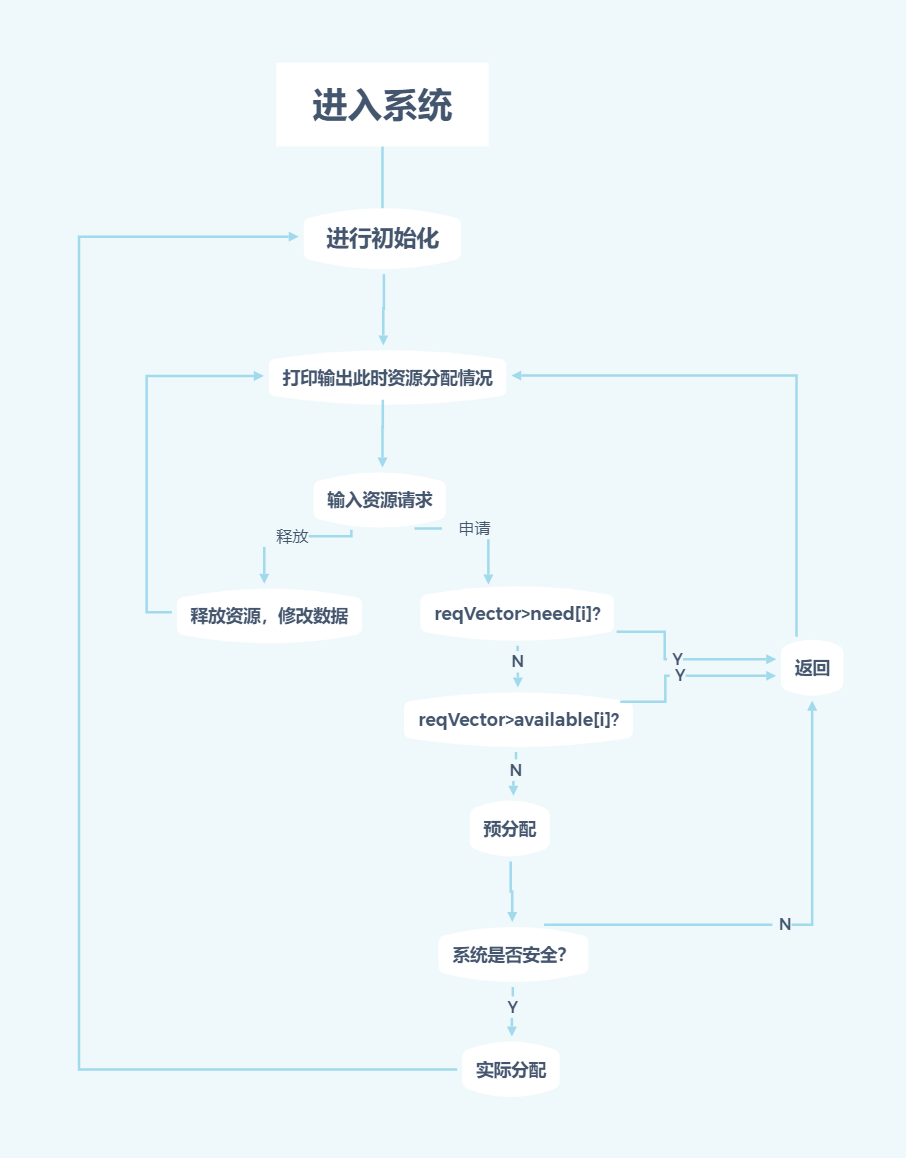
# 实验平台

本实验采用java语言模拟银行家算法，所用平台为：

Windows 10 JDK13.0 Intellj IDEA

未使用java标准库以外的任何第三方库。

# 算法流程



# 详细设计

程序随机生成一定数量的进程以及进程对所有资源的需求总数，并随机生成系统的资源总数。然后用户手动从控制台进行进程调度，如用户决定让1号进程申请某个资源，则用户在控制台输入申请数量，程序依据当前资源分配情况决定是否可以进行分配，分配成功则更改资源分配情况表，然后给出安全序列，分配不成功则对用户进行提示，重新输入资源请求。

首先定义用到的一些工具：random用于随机生成进程对资源数的请求，scanner用于从控制台读取用户手工输入的进程调度。

**static int** *processNum* = 5;*//设定进程数量***static int** *resourceTypesNum* = 5; *//设定资源种类数***static int** *resourceAvailableQuantityStart* = 9;  
**static int** *resourceAvailableQuantityEnd* = 13; *//设定初始化系统拥有每种资源的数量范围***static int** *maxNeedPerResource* = 8; *//设定每个进程最多需求每种资源的数量*

一些基本信息可以在这里更改，比如进程数量、资源种类、系统初始化资源数量的范围、进程每次请求的资源数量范围等等。注意系统资源总量一定要大于每个进程的最大请求数量。

此外还有几个关键的数组和矩阵：

**可利用资源向量 available**：含有 m 个元素的数组，其中每个元素代表一类可用的资源数目 。available [i] = K 表示系统中现有 Rj类资源 K 个。

**最大需求矩阵 maxNeed**: n\*m 矩阵，定义系统中 n 个进程中的每个进程对 m 类资源的最大需求。 简单来说，一行代表一个进程，一列代表一类资源。 Max[i, j] = K表示进程 i 需j类资源的最大数目为 K。

**分配矩阵 allocation**: n\*m 矩阵，定义系统中每类资源当前己分配给每个进程的资源数。allocation [ i,j] = K 表示进程 i 当前已分得 Rj类资源的数目为K。

**需求矩阵 need**: n\*m 矩阵，表示每个进程接下来最多还需要多少资源。 need[i,j] = K表示进程 i 还需要 Rj类资源的数目为K。上述三个矩阵间存在下述关系：

need= maxNeed- allocation

**每次请求操作的进程号reqProcessId**：保存每次用户输入待操作的进程ID

**每次请求操作的进程资源向量reqVector**：保存每次用户输入的资源数组

**标志位isApply**：用来确定每次用户是申请资源还是释放资源，即P/V

**static** List<Integer> *available* = **new** ArrayList<>();  
**static** List<ArrayList<Integer>> *need* = **new** ArrayList<>();  
**static** List<ArrayList<Integer>> *allocation* = **new** ArrayList<>();  
**static** List<ArrayList<Integer>> *maxNeed* = **new** ArrayList<>();  
**static int** *reqProcessId*;  
**static** ArrayList<Integer> *reqVector*;  
**static boolean** *isApply*;

然后是main函数：

**public static void** main(String[] args) {  
 System.***out***.println(**"-------------------------------------------------"**);  
 System.***out***.println(**"- -"**);  
 System.***out***.println(**"- 银行家算法演示 -"**);  
 System.***out***.println(**"- @物联网1802白明强 -"**);  
 System.***out***.println(**"- -"**);  
 System.***out***.println(**"-------------------------------------------------"**);  
 *init*();*//初始化各类资源  
 printStatus*();  
 **while**(**true**){  
 *input*();  
 **if**(*isApply*){  
 *handleAllocation*();  
 }**else**{  
 *releaseResource*();  
 }  
 System.***out***.println(**"---------------------------------------------"**);  
 *printStatus*();  
 System.***out***.println(**"---------------------------------------------"**);  
 }  
}

调用init函数初始化各种数据结构，简单来说就是调用random对象随机生成数据填充available、maxNeed、need等数组，初始已分配矩阵allocation全部置0。printStatus函数打印出当前各数据结构的内容到显示台，方便观察分配情况。

然后用一个while循环包裹主逻辑：

调用input函数，提示用户进行选择释放还是分配资源，并更改标志位，然后输入reqVector向量。input执行完毕后，依据isApply标志位判断，如果是释放资源比较简单，只要将reqVector与available相应的行相加，并将allocation相应的行减去对应数目的资源即可。如果是申请资源，那么进入handleAllocation函数。

**public static void** handleAllocation() {  
 **if**(!*checkAboveEqualZero*(*doMathForList*(**'-'**,*need*.get(*reqProcessId*),*reqVector*))  
 ||!*checkAboveEqualZero*(*doMathForList*(**'-'**,*available*,*reqVector*))){  
 System.***out***.println(**"错误！您所申请的资源数目超过了该进程宣称需要的最大值！"**);  
 }**else**{  
 ArrayList<Integer> newAvaliable;  
 ArrayList<ArrayList<Integer>> newAllocation = **new** ArrayList<>(*allocation*);  
 ArrayList<ArrayList<Integer>> newNeed = **new** ArrayList<>(*need*);  
  
 newAvaliable = *doMathForList*(**'-'**,*available*, *reqVector*);  
 newAllocation.set(*reqProcessId*,*doMathForList*(**'+'**,*allocation*.get(*reqProcessId*),*reqVector*));  
 newNeed.set(*reqProcessId*,*doMathForList*(**'-'**,*need*.get(*reqProcessId*),*reqVector*));  
  
 **if**(*isSafe*(newAvaliable,newAllocation,newNeed)){  
 *available* = *doMathForList*(**'-'**,*available*, *reqVector*);  
 *allocation*.set(*reqProcessId*,*doMathForList*(**'+'**,*allocation*.get(*reqProcessId*),*reqVector*));  
 *need* = newNeed;  
 System.***out***.println(**"分配资源成功！"**);  
 }**else**{  
 System.***out***.println(**"分配资源失败！"**);  
 }  
 }  
}

此处为了代码简洁复用，我们定义了两个工具函数，*doMathForList和checkAboveEqualZero，*前者的作用是输入符号以及两个行向量，可以对这两个行向量进行加减运算，返回一个新的结果对象。后者的作用是判断一个行向量是否全部大于等于0。

handleAllocation首先检查申请资源数量是否超过进程宣称的最大数量，然后检查系统是否剩余足够资源对其进行分配，如果有一个不满足则拒绝分配。当满足这两个情况时，说明当前系统剩余资源数量可以满足进程的此次请求，但是银行家算法要求，必须分配后系统没有进入不安全状态才能进行分配，所以接下来需要对此次分配进行安全性检查。如果安全检查通过，则真正分配给进程资源，如果安全检查没有通过，则拒绝此次分配请求。我们假设进行了分配，得到分配之后的newAvailable、newAllocation、newNeed三个数组，用这三个数组作为参数调用isSafe函数进行安全性检查。

**public static boolean** isSafe(ArrayList<Integer> newAvailable,ArrayList<ArrayList<Integer>> newAllocation,ArrayList<ArrayList<Integer>> newNeed) {  
 ArrayList<Integer> work = **new** ArrayList<>(newAvailable);  
 ArrayList<Integer> safe = **new** ArrayList<>();  
 **for** (**int** i = 0; i < *processNum*; i++) {  
 **boolean** contains = safe.contains(i);  
 **boolean** canAlloc = *checkAboveEqualZero*(*doMathForList*(**'-'**, work,newNeed.get(i)));  
 **if** (!contains && canAlloc) {  
 safe.add(i);  
 work = *doMathForList*(**'+'**, work, newAllocation.get(i));  
 i = -1;  
 } **else** {  
 **if** (safe.size() == *processNum*) {  
 System.***out***.println(**"安全序列为："**);  
 System.***out***.println(safe);  
 **return true**;  
 }  
 }  
 }  
 System.***out***.println(**"会进入不安全状态！分配失败！"**);  
 **return false**;  
}

isSafe函数内部定义work向量，初始化与newAvailable相等。然后定义一个safe向量保存系统的安全序列，初始为空。

每次从进程队列中挑选一个满足这样的条件的进程：

1.进程不在安全队列safe中

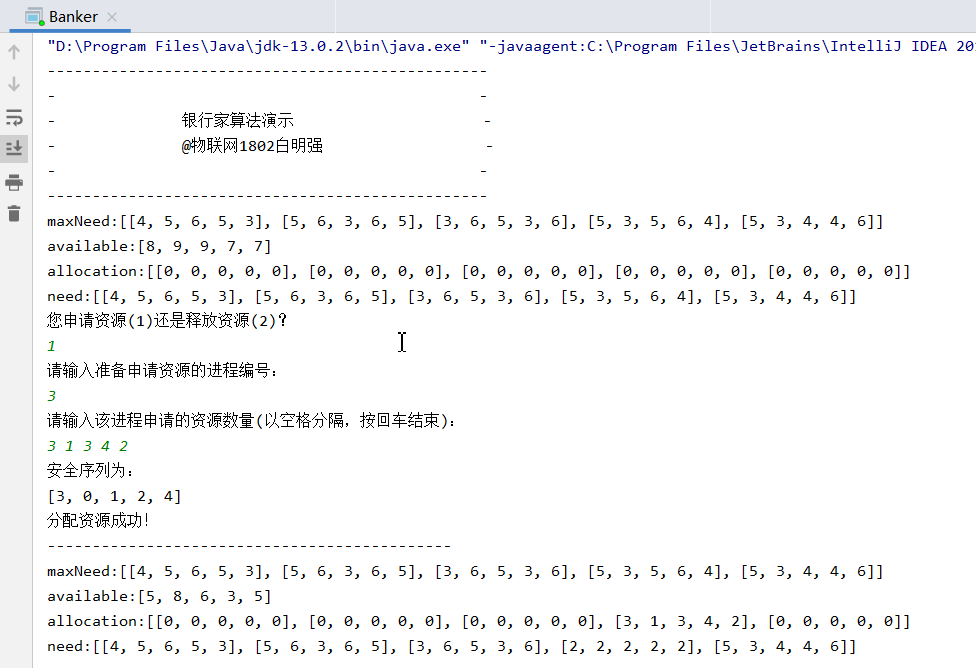
2.进程所剩余未被分配的资源小于系统剩余资源总量(也就是newAvailable-newNeed的对应行全部大于等于0)

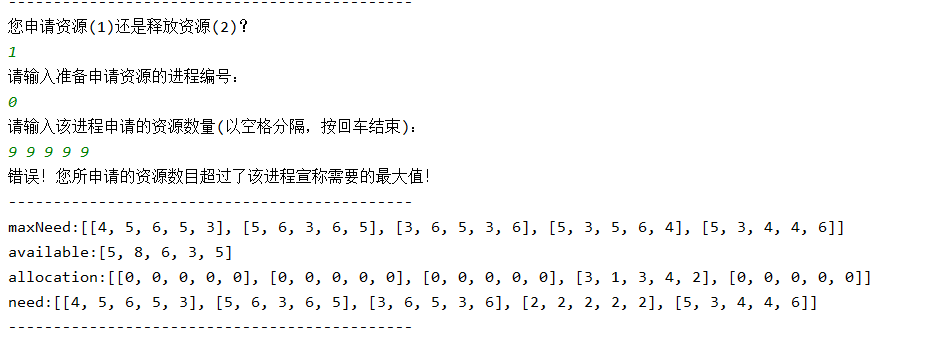
假设该进程被分配了资源然后运行完毕，所以需要释放该进程占有的资源，修改对应的分配数据结构，将当前进程加入安全序列。然后检查安全序列是否已经加入了全部的进程，如果是则找到了一个安全推进序列，isSafe返回true。如果安全序列未满，则继续从进程队列找寻满足以上要求的进程，直到安全队列满，如果找不到这样的进程，则此次分配会使系统进入不安全状态，isSafe函数返回false。

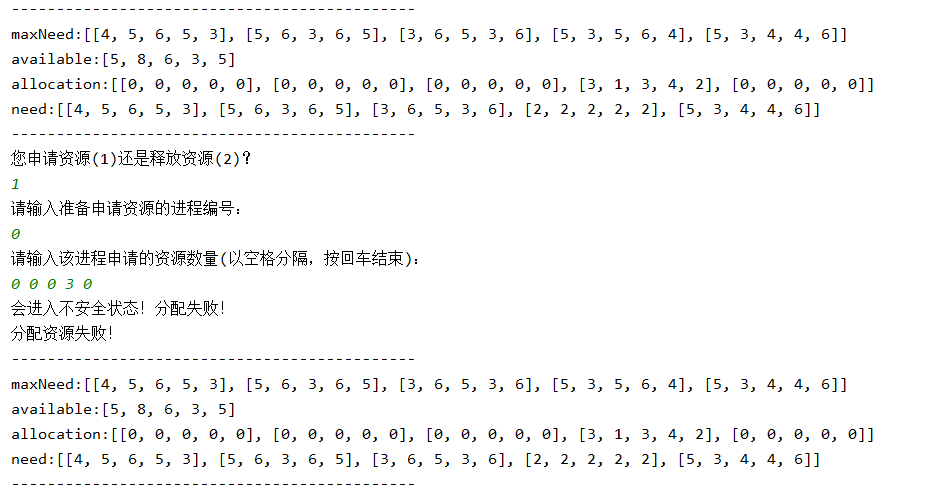
然后回到了main函数中的循环，打印出新的系统资源分配情况后，开启新一轮循环。程序正常推进，算法演示完毕。

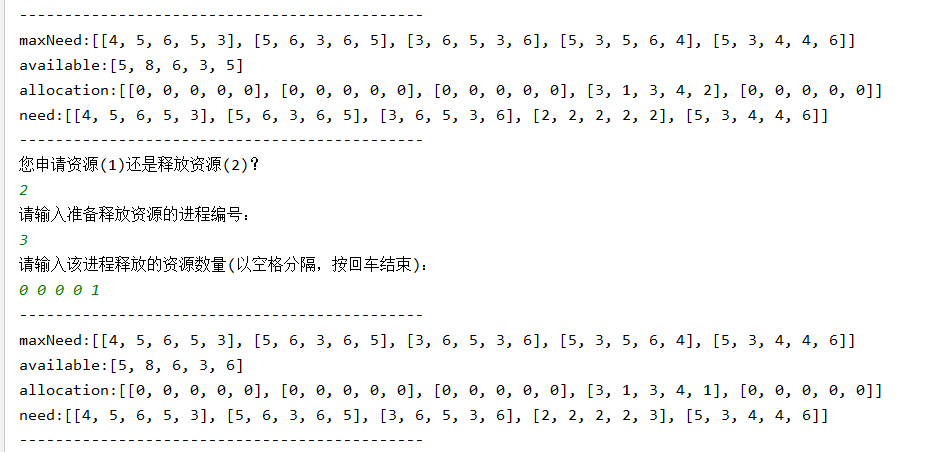
# 结果演示

正常申请：



超额申请：

不安全被拒：

释放资源：

# 完整代码

**import** java.util.\*;  
  
*/\*\*  
 \** ***@author*** *7zq12lvm  
 \** ***@create*** *2020-05-13 16:42  
 \*/***public class** Banker {  
  
 **static** Random *random* = **new** Random();  
 **static** Scanner *scanner* = **new** Scanner(System.***in***);  
 **static int** *processNum* = 5;*//设定进程数量* **static int** *resourceTypesNum* = 5; *//设定资源种类数* **static int** *resourceAvailableQuantityStart* = 7;  
 **static int** *resourceAvailableQuantityEnd* = 10; *//设定初始化系统拥有每种资源的数量范围* **static int** *maxNeedPerResource* = 4; *//设定每个进程最多需求每种资源的数量* **static** List<Integer> *available* = **new** ArrayList<>();  
 **static** List<ArrayList<Integer>> *need* = **new** ArrayList<>();  
 **static** List<ArrayList<Integer>> *allocation* = **new** ArrayList<>();  
 **static** List<ArrayList<Integer>> *maxNeed* = **new** ArrayList<>();  
 **static int** *reqProcessId*;  
 **static** ArrayList<Integer> *reqVector*;  
 **static boolean** *isApply*;  
  
 **public static void** main(String[] args) {  
 System.***out***.println(**"-------------------------------------------------"**);  
 System.***out***.println(**"- -"**);  
 System.***out***.println(**"- 银行家算法演示 -"**);  
 System.***out***.println(**"- @物联网1802白明强 -"**);  
 System.***out***.println(**"- -"**);  
 System.***out***.println(**"-------------------------------------------------"**);  
  
  
 *init*();*//初始化各类资源  
  
 printStatus*();  
  
 **while**(**true**){  
 *input*();  
 **if**(*isApply*){  
 *handleAllocation*();  
 }**else**{  
 *releaseResource*();  
 }  
  
 System.***out***.println(**"---------------------------------------------"**);  
 *printStatus*();  
 System.***out***.println(**"---------------------------------------------"**);  
 }  
 }  
  
 **public static void** handleAllocation() {  
 **if**(!*checkAboveEqualZero*(*doMathForList*(**'-'**,*need*.get(*reqProcessId*),*reqVector*))  
 ||!*checkAboveEqualZero*(*doMathForList*(**'-'**,*available*,*reqVector*))){  
 System.***out***.println(**"错误！您所申请的资源数目超过了该进程宣称需要的最大值！"**);  
 }**else**{  
 ArrayList<Integer> newAvaliable;  
 ArrayList<ArrayList<Integer>> newAllocation = **new** ArrayList<>(*allocation*);  
 ArrayList<ArrayList<Integer>> newNeed = **new** ArrayList<>(*need*);  
  
 newAvaliable = *doMathForList*(**'-'**,*available*, *reqVector*);  
 newAllocation.set(*reqProcessId*,*doMathForList*(**'+'**,*allocation*.get(*reqProcessId*),*reqVector*));  
 newNeed.set(*reqProcessId*,*doMathForList*(**'-'**,*need*.get(*reqProcessId*),*reqVector*));  
  
 **if**(*isSafe*(newAvaliable,newAllocation,newNeed)){  
 *available* = *doMathForList*(**'-'**,*available*, *reqVector*);  
 *allocation*.set(*reqProcessId*,*doMathForList*(**'+'**,*allocation*.get(*reqProcessId*),*reqVector*));  
 *need* = newNeed;  
 System.***out***.println(**"分配资源成功！"**);  
 }**else**{  
 System.***out***.println(**"分配资源失败！"**);  
 }  
 }  
 }  
  
 **public static boolean** isSafe(ArrayList<Integer> newAvailable,ArrayList<ArrayList<Integer>> newAllocation,ArrayList<ArrayList<Integer>> newNeed) {  
 ArrayList<Integer> work = **new** ArrayList<>(newAvailable);  
  
 ArrayList<Integer> safe = **new** ArrayList<>();  
  
  
 **for** (**int** i = 0; i < *processNum*; i++) {  
 **boolean** contains = safe.contains(i);  
 **boolean** canAlloc = *checkAboveEqualZero*(*doMathForList*(**'-'**, work,newNeed.get(i)));  
 **if** (!contains && canAlloc) {  
 safe.add(i);  
 work = *doMathForList*(**'+'**, work, newAllocation.get(i));  
 i = -1;  
 } **else** {  
 **if** (safe.size() == *processNum*) {  
 System.***out***.println(**"安全序列为："**);  
 System.***out***.println(safe);  
 **return true**;  
 }  
 }  
  
 }  
 System.***out***.println(**"会进入不安全状态！分配失败！"**);  
 **return false**;  
  
 }  
  
 **public static void** releaseResource() {  
 *available* = *doMathForList*(**'+'**,*available*,*reqVector*);  
 *allocation*.set(*reqProcessId*,*doMathForList*(**'-'**,*allocation*.get(*reqProcessId*),*reqVector*));  
 *need*.set(*reqProcessId*,*doMathForList*(**'+'**,*need*.get(*reqProcessId*),*reqVector*));  
 }  
  
 **public static** ArrayList<Integer> doMathForList(**char** type,List<Integer> first,List<Integer> second){  
 ArrayList<Integer> result = **new** ArrayList<>();  
 **if**(type == **'+'**){  
 **for** (**int** i = 0; i < *resourceTypesNum* ; i++) {  
 result.add(first.get(i)+second.get(i));  
 }  
 }  
 **if**(type == **'-'**) {  
 **for** (**int** i = 0; i < *resourceTypesNum*; i++) {  
 result.add(first.get(i) - second.get(i));  
 }  
 }  
 **return** result;  
 }  
  
 **public static boolean** checkAboveEqualZero(List<Integer> l){ *//检查数组的一行是否全部大于0* **for** (Integer integer : l) {  
 **if**(integer<0){  
 **return false**;  
 }  
 }  
 **return true**;  
 }  
  
  
 **public static void** init(){  
 *//系统资源数量available  
 random*.ints(*resourceTypesNum*,*resourceAvailableQuantityStart*,*resourceAvailableQuantityEnd*).forEach(*available*::add);  
 *//生成进程需求矩阵maxNeed* **for** (**int** i = 0; i < *processNum* ; i++) {  
 ArrayList<Integer> temp = **new** ArrayList<>();  
 **for** (**int** j = 0; j < *resourceTypesNum*; j++) {  
 temp.add(*random*.nextInt(*maxNeedPerResource*)+3);  
 }  
 *maxNeed*.add(temp);  
 }  
  
 *//生成need矩阵，保存当前状态下进程需要的资源数量，初始化为maxNeed* **for** (**int** i = 0; i < *processNum* ; i++) {  
 *need*.add(**new** ArrayList<>());  
 **for** (**int** j = 0; j < *resourceTypesNum* ; j++) {  
 *need*.get(i).add(*maxNeed*.get(i).get(j));  
 }  
 }*//深拷贝  
  
 //已经分配的资源数量* **for** (**int** i = 0; i < *processNum*; i++) {  
 ArrayList<Integer> temp = **new** ArrayList<>();  
 **for** (**int** j = 0; j < *resourceTypesNum*; j++) {  
 temp.add(0);  
 }  
 *allocation*.add(temp);  
 }  
 }  
  
 **public static void** input(){  
  
 System.***out***.println(**"您申请资源(1)还是释放资源(2)？"**);  
 **int** choice=*scanner*.nextInt();  
 **if**(choice==1){  
 *isApply*=**true**;  
 System.***out***.println(**"请输入准备申请资源的进程编号："**);  
 *reqProcessId* = *scanner*.nextInt();*//请求资源的进程编号* System.***out***.println(**"请输入该进程申请的资源数量(以空格分隔，按回车结束)："**);  
 *reqVector* = **new** ArrayList<>();  
 **for** (**int** i = 0; i < *resourceTypesNum*; i++) {  
 *reqVector*.add(*scanner*.nextInt());  
 }  
 }**else if**(choice==2){  
 *isApply*=**false**;  
 System.***out***.println(**"请输入准备释放资源的进程编号："**);  
 *reqProcessId* = *scanner*.nextInt();*//释放资源的进程编号* System.***out***.println(**"请输入该进程释放的资源数量(以空格分隔，按回车结束)："**);  
 *reqVector* = **new** ArrayList<>();  
  
 **for** (**int** i = 0; i < *resourceTypesNum*; i++) {  
 *reqVector*.add(*scanner*.nextInt());  
 }  
 }**else**{  
 System.***out***.println(**"输入有误，请重新输入！"**);  
 *input*();  
 }  
 }  
 **public static void** printStatus(){  
 System.***out***.print(**"maxNeed:"**);  
 System.***out***.println(*maxNeed*);  
  
 System.***out***.print(**"available:"**);  
 System.***out***.println(*available*);  
 System.***out***.print(**"allocation:"**);  
 System.***out***.println(*allocation*);  
 System.***out***.print(**"need:"**);  
 System.***out***.println(*need*);  
 }  
  
}

# 实验总结

通过本次实验，深入体会了银行家算法在解决操作系统最核心问题--死锁方面的优点，进一步加深了对操作系统在计算机世界承上启下作用的理解，也提高了动手能力，培养了独自解决问题的能力。例如，java特有的强面向对象的特性，只复制引用，导致无法很好地对数据进行复制，所以要手动对矩阵每一个数据项进行值复制来解决问题。又例如，实验中要进行行向量的加减，java没有提供此类函数，只能自己写一个函数来解决。以上种种，又加深了对java语言的理解。总之，本次实验收获颇丰。