一.实验目的:

银行家算法是避免死锁的一种重要方法，能够有效的在资源分配过程中，对系统的安全性进行检测。

通过银行家算法设计和实现,可以加深对思索地理解，掌握思索地预防，避免，检测和解除的理解，掌握思索地预防，避免，检测和解除的基本原理，重点掌握思索地避免方法—银行家算法。初步具有研究，设计，编制和调试操作系统模块的能力。

二.设计原理

（1）可用资源向量Available[m]

m为系统中资源种类数,Available[j]=k表示系统中第j类资源数为k个，

（2）最大需求矩阵Max[n][m]

N为系统中进程数，Max[i][j]=k表示进程i对j类资源的最大需求数为k。

（3）分配矩阵Allocation[n][m]

Allocation[i][j]=k表示进程i已分得j类资源的数目为k个。

（4）需求矩阵Need[n][m]

Need[i][j]=k表示进程i还需要j类资源k个。

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

三.算法介绍

假设在进程并发执行时进程i提出请求j类资源k个后,表示为Requesti[j]=k,系统按照下述步骤进行安全检验:

如果Requesti<=Needi,则继续以下检查,否则显示需求申请超过最大需求值的错误。

如果Requesti<=Available则继续以下检查,否则显示系统无足够资源，Pi阻塞等待。

系统试探把要求的资源分配给进程i并修改有关数据结构的值：

Available=Available-Requesti；

Allocationi=Allocationi+requesti；

Needi=Needi-Requesti；

系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态，若安全，才正式将资源分配给进程i，以完成本次分配；否则将试探分配作废，恢复原来的资源分配状态，让进程Pi等待。

四。安全性算法

（1）设置完成标志向量Finish[n]。

初始化：Finish[i]=false表示i进程尚未完成。

（2）从进程集合设置Work[m]表示系统可提供给进程的的各类资源数目。初始值Work=Available。

（3）当进程i获得资源后可顺利执行直到完成，并释放出分配给他的资源，表示如下：

Work=work+Allocationi；Finish[i]=true；

转至（2）

（4）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。

五.代码实现

该代码是一个基于Java Swing图形界面实现的银行家算法模拟程序，用于演示操作系统中的进程管理。程序主要包括初始化资源、进程以及进行安全性检测和资源分配等功能。

5.1主要类和方法

main2类:

静态变量定义:

tesp: 记录当前正在初始化的进程序号。

KINDS: 资源种类数量。

resource: 总资源数数组。

ProcessCount: 进程数量。

team: 存储进程信息的ArrayList。

avaliable: 当前可分配资源数组。

jFrame: 主窗口JFrame实例。

5.2 GUI组件:

本实验使用了JFrame,JTextField,JLable,JButton,JMenu,JMenubar,JMenuitem

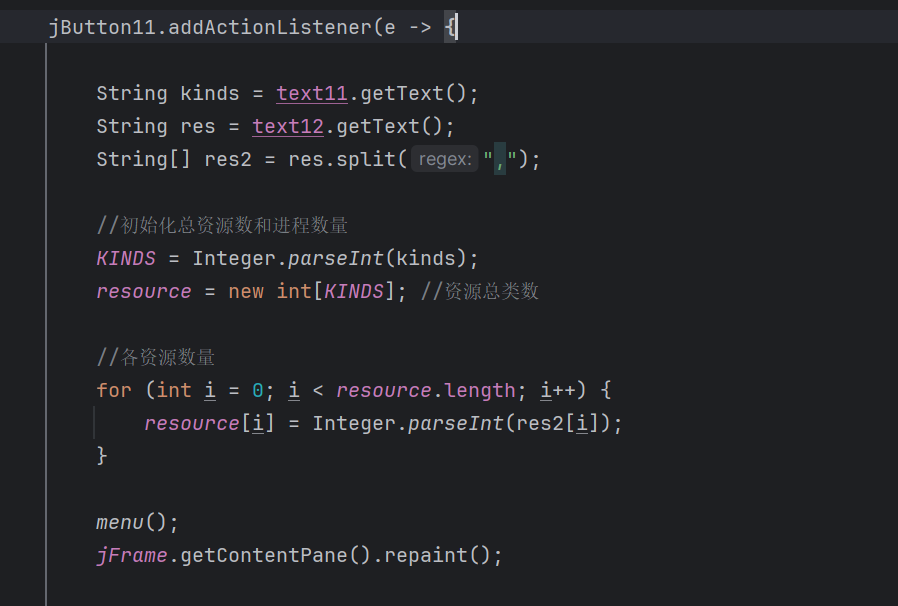


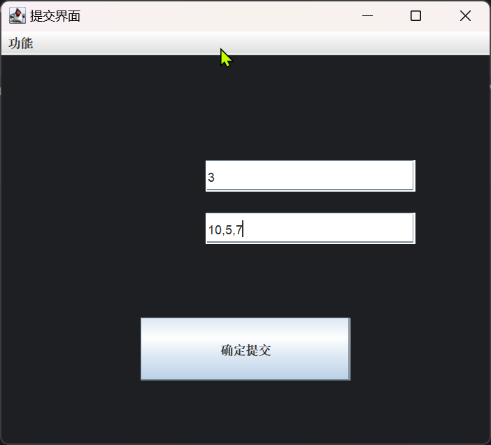
方法概览:

5.3 InitAllResource方法:

功能：初始化总资源数。创建并展示一个Swing界面供用户输入资源种类数量和每种资源的具体数量。

用户通过界面输入后，点击“确定提交”按钮，程序解析输入并初始化资源数组。





5.4 InitProcess方法:

功能：初始化进程信息。继续使用图形界面让用户输入进程的个数，并为每个进程输入名称、最大需求资源和已分配资源。

用户逐个输入进程信息，完成后，程序会根据输入构建进程对象并加入到team列表中。





5.5 检查系统是否处于安全状态、分配资源给进程、安全状态检测：判断当前系统资源分配状态是否安全，即是否存在一种资源分配序列，使得所有进程都能依次完成执行而不发生死锁。

输入与输出

输入：无直接输入参数，依赖于全局变量team(进程列表)、KINDS(资源种类数)、avaliable(当前可用资源)、ProcessCount(进程总数)等。

输出：布尔值，true表示当前状态安全，false表示当前状态不安全。

核心逻辑步骤

前置检查：

验证team是否已经初始化，若未初始化则提示用户并返回false。

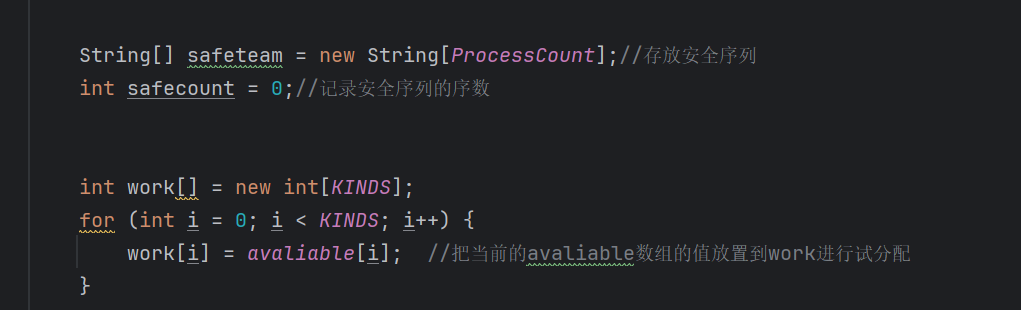
检查avaliable数组中的元素，确保没有负数，表示资源分配不能超过总量，否则提示错误并重置相关状态。

初始化安全序列尝试：

创建一个长度等于ProcessCount的字符串数组safeteam，用于存储安全序列。

复制当前avaliable数组到work数组，用于模拟分配资源。

寻找安全序列：



通过循环遍历team中的每个进程，尝试为每个进程分配资源。

判断team.get(index).finshined标记，决定是否跳过已检查的进程。

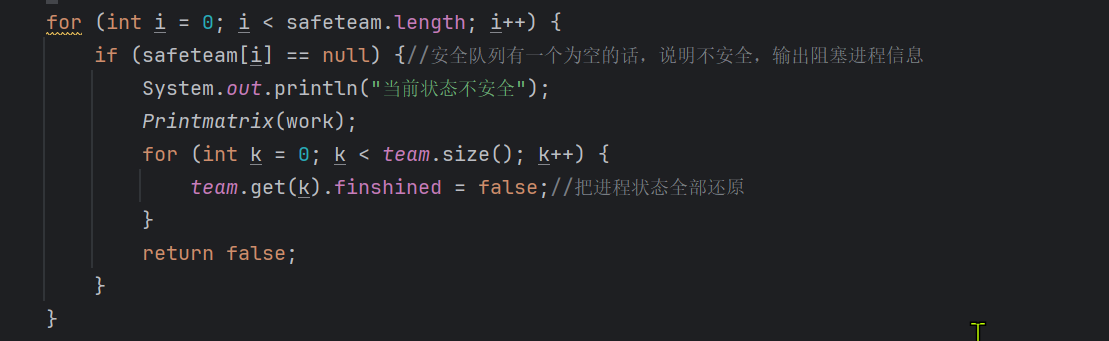
对每个进程，比较其需求资源needs与当前可分配资源work，如果满足分配条件，则标记该进程为已分配状态，并更新work数组。

将分配成功的进程添加到安全序列safeteam中，更新安全序列计数safecount。



安全状态判断：

遍历safeteam，如果发现任何位置为空，表明未找到安全序列，因此状态不安全，输出相关信息并还原进程状态后返回false。



若安全序列完整，输出安全序列，表示状态安全。

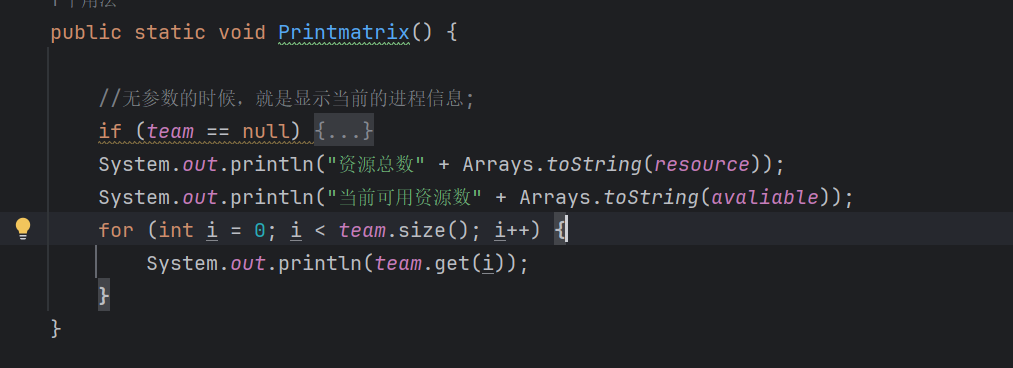
资源实际分配（假设安全）：

仅当所有进程的needs均为0，表示所有进程资源需求都已满足，执行实际的资源分配操作，即将分配出去的资源从team.get(index).allocation移除并累加到avaliable中，同时重置进程的分配状态。



最后，无论是否进行了实际分配，都会还原所有进程的finshined标记为false，以保持状态的一致性。

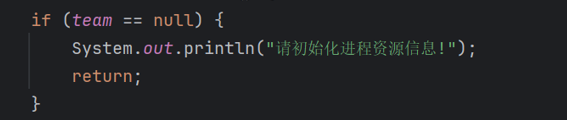
5.6 显示当前资源和进程状态等。



5.7 银行家算法:



检查初始化状态:



首先检查team是否为null，这是确保之前已经完成了资源和进程的初始化。如果没有初始化，程序会打印提示信息并直接返回。

遍历team列表，查找与用户输入名字相匹配的进程。

一旦找到匹配的进程，就进入资源请求分配的逻辑。

请求资源分配:

根据资源种类数KINDS创建一个整型数组request来存储用户请求的每种资源的数量。

通过循环，逐个询问并获取用户对每种资源的请求量。

资源请求合理性检查:



内部有两个循环，分别检查两个条件：

请求的资源是否超过了该进程当前的需求量。

请求的资源是否超过了系统当前可分配的资源量。

如果任一条件不满足，程序会打印错误信息并返回，阻止资源分配。

尝试分配资源:

如果前两步的检查都通过，调用TryAllcotion(i, request)方法尝试分配资源。这里i是匹配到的进程在team列表中的索引，request是用户请求的资源数组。

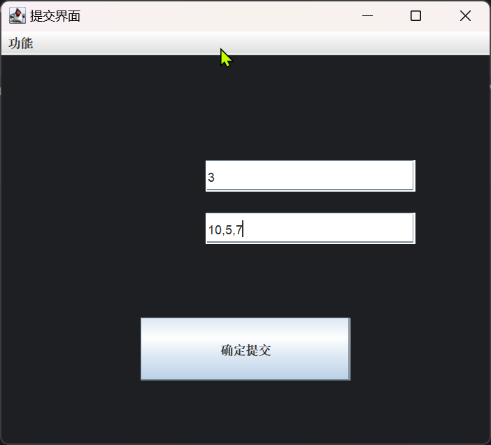
分配后，直接返回，结束方法执行。

六 .运行图:

可视化GUI组件:



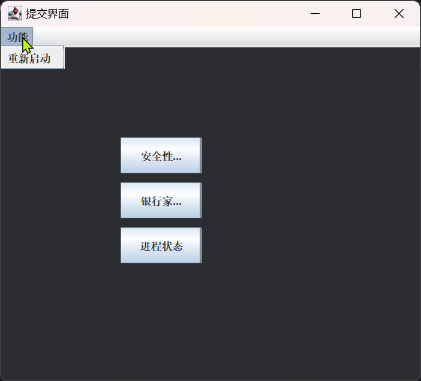
初始化总资源数量:



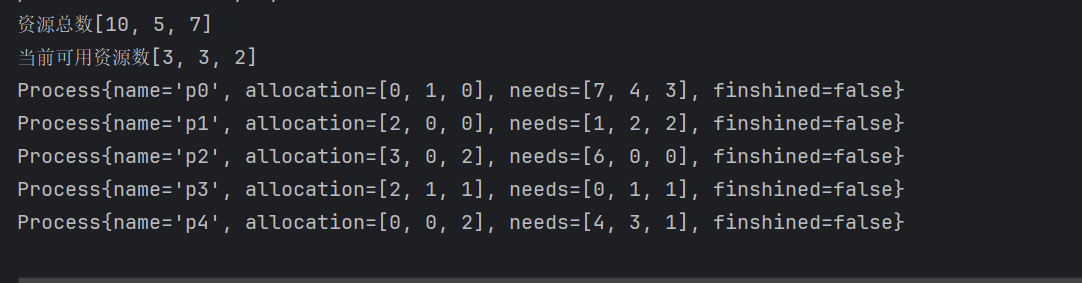
初始化各进程数量:



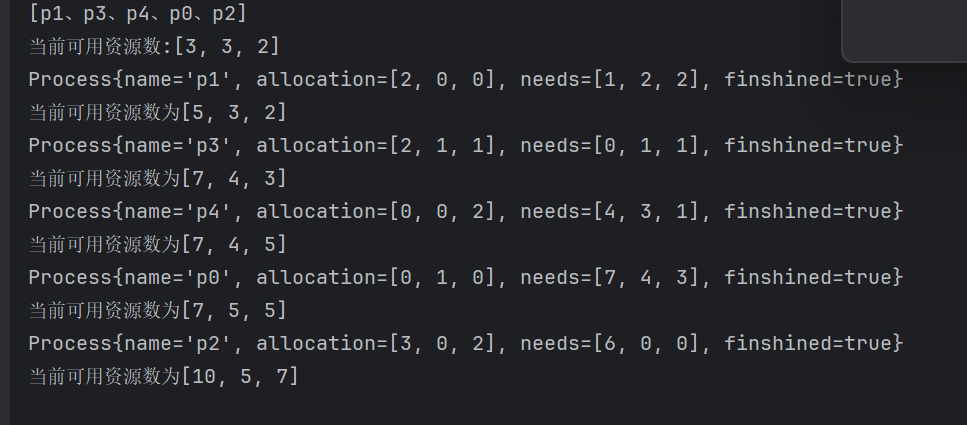
控制界面:



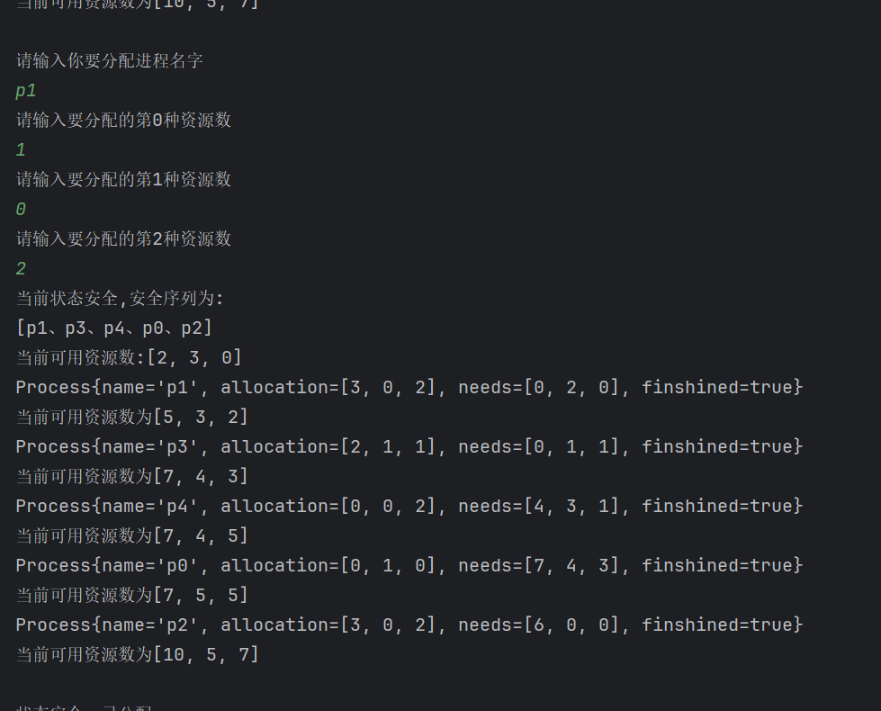
按钮1:



按钮2:



按钮3:



七．实验心得：

通过这次实验，使我加深了对银行家算法思想的理解和操作系统对资源死锁的预防措施。