# 实验二：银行家算法

|  |  |
| --- | --- |
| 实验名称 | 实验二：银行家算法的模拟实现 |
| 姓名学号 | 高丽 2017302207 |
| 1. **实验目的** 2. 用C++模拟实现银行家算法，以更好地掌握该算法的原理、理解其如何避免死锁、理解安全性算法的内容。 3. 同时提高使用编程解决实际问题的能力。 | |
| 1. **实验内容** 2. 使用语言：C++ 3. 数据结构：   我的设计主要包含五个数组：Available[]可利用资源向量、Max[][]最大需求矩阵、Allocation[][]分配矩阵、Need[][]需求矩阵、Request[][]请求向量。  Available[j]=K表示系统有类资源K个；Max[i][j] = K，表示进程i需要资源最大数目为K；Allocation[i][j] = K，表示进程i当前已分得类资源得数目为K；Need[i][j] = K，表示进程i还需要类资源K个方能完成其任务；Request[i][j]=K，表示进程i请求K个类资源。  定义两个宏MAX\_PROCESSES、MAX\_RESOURCES，分别表示最大进程数和最大资源数，值分别为20、15。这两个宏用于所有数组的初始大小。  定义两个变量m资源数、n进程数，用于用户自己设定模拟的进程数量和资源数量，m和n的值不得大于两个宏的值。   1. 流程： 2. 总体思路：（流程图如下）      1. 具体实现：  * 由以下几个函数构成：   bool SafetyCheck ();//安全检查函数  void PrintSafety() ;//输出查看函数  void Init();//初始化Max、Allocation、Need、Available数组  void PRequest ();//进程申请资源函数  void Banker();//算法主体函数   * 具体说明SafetyCheck()安全检查函数（流程图如下）      * 详细说明流程： * 第一步：输入进程数量和资源数量，初始化Max、Allocation、Need、Available四个数组并作安全检查   (i) 输入n进程数、m资源数，检查是否大于预先规定的最大值。  (ii) 调用Init()函数初始化：首先输入Max[P][R]、Allocation[P][R]、Available[R]，并计算需求矩阵Need[P][R]，其次，设置两个数组Safetyqueue[]安全序列、workandallocation[][]，调用安全检查函数，如果为true则调用PrintSafety()输出，否则，重新输入矩阵值。   * 第二步：进程申请资源，使用银行家算法（这一步是一个循环，仅当申请的进程号p为-1时，退出循环，结束算法模拟过程）   (i) 输入进程号p，判断其范围是否在在[0,n)范围内，若不在重新输入，否则继续输入p请求资源的数量即Request[p][r]。  (ii) 如果p为-1，退出循环，即退出模拟过程。  (iii) 调用银行家算法Banker()函数：判断对所有属于[0,m)的资源r，Request[p][r] > Available[r]或者Request[p][r] > Need[p][r]是否成立，成立则申请资源失败，否则执行下一步；  (iv)修改以下数组： Available[r] -= Request[p][r];  Allocation [p][r]+= Request[p][r];  Need[p][r] -= Request[p][r];  该步对所以r∈[0,m)均操作。  (v) 设置两个数组Safetyqueue[]安全序列、workandallocation[][]，调用SafetyCheck ()，如果为true则输出Max矩阵、Need矩阵、Allocation矩阵，否则，试探作废：Available[r] += Request[p][r];  Allocation[p][r] -= Request[p][r];  Need[p][r] += Request[p][r]。  该步对所以r∈[0,m)均操作。 | |
| 1. **实验结果**（红色箭头为输入）   可自由输入进程数（不大于20）和资源数（不大于15）。  ① 输入进程数为5，资源数为3。输入Max 、Allocation。这里模拟书上P113页例子。    输出得到计算的Need值  ② 输入Available。    ③进程1请求资源，数量为{1 0 2}，结果如下。    ④进程4请求资源，数量为{3 3 0}    ⑤进程0请求资源，数量为{0 2 0}    ⑥进程0重新请求资源，数量改为{0 1 0}    ⑤输入进程-1，结束模拟。 | |
| 1. **结果分析**   初始化时，输入设定进程数为5，资源数为3。  且输入如下。Need为计算结果，其他为输入值。   |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 资源情况/进程 | Max | | | Allocation | | | Need | | | Available | | | | A | B | C | A | B | C | A | B | C | A | B | C | |  | 7 | 5 | 3 | 0 | 1 | 0 | 7 | 4 | 3 | 3 | 3 | 2 | |  | 3 | 2 | 2 | 2 | 0 | 0 | 1 | 2 | 2 |  | | | |  | 9 | 0 | 2 | 3 | 0 | 2 | 6 | 0 | 0 | |  | 2 | 2 | 2 | 2 | 1 | 1 | 0 | 1 | 1 | |  | 4 | 3 | 3 | 0 | 0 | 2 | 4 | 3 | 1 |   先进行安全性检查，得到如下结果，安全序列为{1 3 0 2 4}。    ①接着，进程1请求资源，A、B、C数量为{1 0 2}， 继续安全性检查，序列为{1 3 0 2 4}，如下图。    这时，输出分配资源后的情况：进程[1]的Allocation值变为原先的{2 0 0}+申请的{1 0 2} ={ 3 0 2 }，Need值也变为{1 2 2}-{1 0 2}={0 2 0}。  此时Available变为{2 3 0}，如下图。    ②进程4请求资源，数量为{3 3 0}, Request[4][r] <=Available[r],所有r∈[0,m)均成立；但是，Request[4][0]=3 > Available[0] =2，故进程[4]阻塞。  ③进程0请求资源，数量为{0 2 0},继续安全性检查，失败，原因：若分配给进程0，则Available变为{2 1 0}，不满足任何进程的需要，所以不安全，拒绝分配资源。  ④进程0重新请求资源，数量为{0 1 0},继续安全性检查，序列为{1 3 0 2 4}，如下图。    这时，输出分配资源后的情况：进程[1]的Allocation值变为原先的{0 1 0}+申请的{0 1 0} ={ 0 2 0}，Need值也变为{7 4 3}-{0 1 0}={7 3 3}。  此时Available变为{2 2 0}，如下图。    以上为对实验结果全部流程的分析。  综上，本次模拟过程完全符合算法要求。 | |