中山大学移动信息工程学院本科生实验报告

（2017 学年春季学期）

课程名称：Operating System　　任课教师：饶洋辉　　批改人(此处为 TA 填写)：

年级+班级	1501	专业（方向）	移动信息工程
学号	15352006	姓名	蔡丽芝
电话	13538489980	Email	314749816@qq.com
开始日期	2017/6/6	完成日期	2017/6/6

1. 实验目的

实现银行家算法模拟操作系统管理分配资源

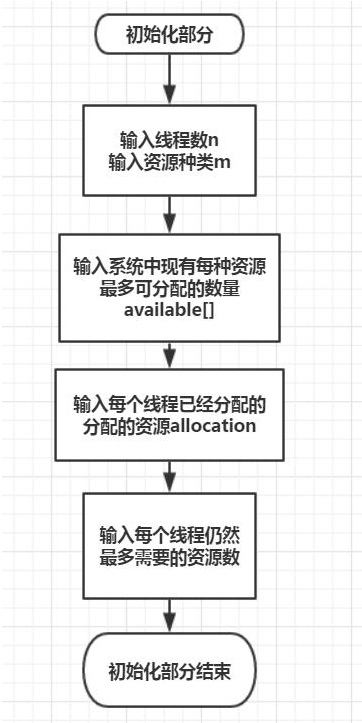
2. 实验过程

（一）算法简述

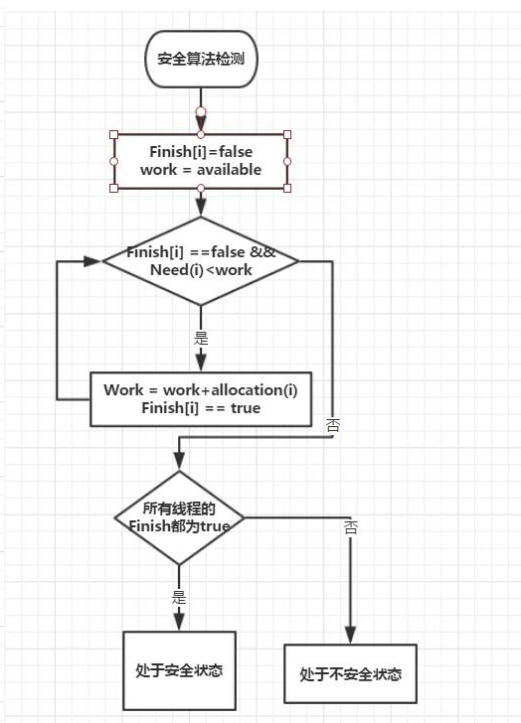
银行家算法，现实生活中，客户贷款的最大需求量不得超过银行家现有的可利用资金，虽然客户可进行分期贷款，但是贷款的总数不能大于最大需求量，即使当银行家现有可利用的资金无法满足客户的贷款数额，也可以推迟支付，银行家总能在有限的时间内使客户得到贷款。 当客户得到所需的资金后，一定要在有限规定的时间内归还所有的资金，在上述的描述过程中就是把操作系统看成是银行家，操作系统管理的资源看成是银行家管理的资金，进程向操作系统请求分配资源看成是用户向银行家贷款。

（二）伪代码或流程图

初始化部分



安全检测部分



（三）具体实现

初始化部分

```
// 初始化
cin >> n >> m;
int available[m];                    // 表示一开始系统现有每一个资源可分配的数量
int allocation[n][m];                // 表示进程已经已经分配的资源数组
int need[n][m];                      // 表示进程的需求数组
bool Finish[n];                      // 表示进程是否已经被分配,
int work[m];                         // 表示系统任意时刻每一个资源可分配的数量
int safe[n];                         // 用于储存已分配资源线程的数组
for(int i = 0; i < m; i++)
   cin >> available[i];
for(int i = 0; i < n; i++)
  for(int j = 0; j < m; j++)
     cin >> allocation[i][j];
for(int i = 0; i < n; i++)
  for(int j = 0; j < m; j++)
     cin >> need[i][j];
```

解释在注释中

安全检测部分

```
for(int i = 0; i < n; i++)                  // 未进行任何分配操作前，将finish数组初始化为false
    Finish[i] = false;
for(int i = 0; i < m; i++)
    work[i] = available[i];                 // 首先将work数组的值初始化为available的值
int k = 0;                                  // 记录可被分配资源线程的个数，用于判断是否能构成安全序列
```

```
for(int i = 0; i < n; i++)
{
    int right = 0;                                      // 用于记录每个进程满足安全条件的个数
    /*首先第一个条件是进程i没有被分配资源*/
    if(Finish[i] == false)
    {
       /*遍历线程i的每一个资源的需求量，判断是否满足小于等于系统现有可分配的数量*/
        for(int j = 0; j < m; j++)
        {

           if(need[i][j] <= work[j])
             right = right+1;                           // 如果满足的话，就将right加1
        }

       /* 当right = m，表示进程i所申请的所有资源小于系统现有资源，进程i可得到资源*/
        if(right == m)
        {
            Finish[i] = true;                           //进程i的finish值变为true，已经被分配资源
            safe[k] = i;                                //将i记录到safe数组中，记录安全序列
            k = k +1;                                   // 用于判断是否满足所有的进程都能被安全分配资源
            for(int j = 0; j < m; j++)                  // 进程i完成后，要释放掉已分配的资源allocation
               work[j] = work[j] + allocation[i][j];
            i = -1;                                     //由于要求安全序列是按升序，因此，另i = -1， 从头开始再进行判断
        }
    }

}
```

```
if(k == n)                                              // k = n， 表示所有的线程都满足分配条件，系统处于安全状态。
{
   cout << "Yes ";
   for(int i = 0; i < n-1; i++)
     cout << safe[i] << " ";
   cout << safe[n-1] << endl;
}
else                                                    // 部分线程无法成功获得资源，不安全状态
cout << "No" << endl;
```

3. 实验结果

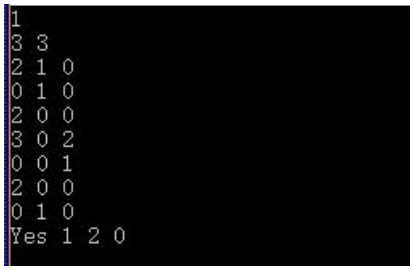
//除了题干中的样例外，请自己模拟两个简单的输入（安全和不安全），并通过计算来解释这两个输入的理论输出是什么，再贴上自己代码实际的输出，最后请截上 Standing 中自己的通过情况

	Allocation	Need	Avaiable
P0	0 1 0	0 0 1	2 1 0
P1	2 0 0	2 0 0	
P2	3 0 2	0 1 0	

理论计算

P0,need 值中并不是每种资源需求量都小于系统现有的资源值，而 P1 满足上述条件，所以系统会先分配资源给 P1，P2 执行完后，释放掉 allocation 的值，此时系统的 available 值为 4 1 0， P2 的各资源需求值小于 available，系统可分配资源给 P2，随后，P2 释放自身资源，available 值变为 7 1 2，同理 P0 也满足上述条件，此时所有线程都可以被分配资源，处于安全状态，且其中一种安全序列为 1 2 0

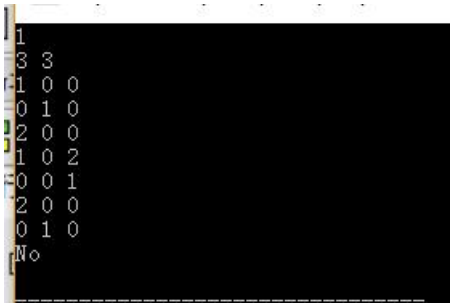
代码输出



	Allocation	Need	Avaiable
P0	0 1 0	0 0 1	1 0 0
P1	2 0 0	2 0 0	
P2	1 0 2	0 1 0	

理论计算

P0,P1,P2 的每个 need 需求都大于 available 值，不满足条件，处于不安全状态



4. 回答问题

➢ 1. 没有安全序列，一定会导致死锁吗，为什么？(参考百度)

答：不存在一个安全序列，不一定会导致死锁。银行家算法所求出的安全序列是以考虑情况最差的条件为前提，因此安全序列一定不会导致死锁。但是，不安全序列不一定会导致死锁，只是表明操作系统分配过程中存在不安全的因素。资源在分配的过程中可能发生变化，可能会存在一种现象，当线程被阻塞的时候，会先释放掉自己的一些资源，这些释放掉的资源加到系统的 work 中，就可能满足后面的线程获取资源，此时并不会出现死锁。比如说条件变量的情况下，线程在符合某些条件的时候，会释放掉一些资源，

然后阻塞自己，这样就能满足上述所说的情况。因此，没有安全序列，不一定会导致死锁。

- 2. 银行家算法实用吗，为什么？
  答：不实用，因为在银行家算法中，每类资源的数量是固定不变的，而且必须事先知道进程所需要的资源数目，而我们是无法提前预知的。

- 3. 查阅并介绍现代操作系统是如何处理死锁的。(参考<<现代操作系统>>和博客)
  答：存在死锁的 4 个必要条件是互斥条件，占有且等待条件，资源不可抢占条件，形成环路条件，如果死锁情况发生，必将满足上述的 4 个条件，缺一不可。所以，可以通过控制这四种条件的发生，来减少死锁现象的发生。通常处理死锁的 4 种策略
  ①采用鸵鸟算法，直接忽略死锁问题，鸵鸟算法的思路就是如果死锁发生的概率很小，并且预防死锁的成本高于死锁发生后的解决成本的时候，直接忽略死锁问题
  ②检测出死锁的地方，并且进行恢复
  检测死锁：
  a.每种类型只有一个资源：构建资源分配图，通过 DFS 的搜索方式，检测图中是否存在环路。
  b.每种类型有多个资源：
  死锁恢复：破坏死锁成立的 4 个条件
  a.将某个资源从它当前的拥有者抢占出来给另一个进程使用，利用抢占恢复
  b.对进程周期性的检测，当检测到死锁的地方，就恢复上一个正常的地方
  c.杀死死锁环路中的另外一个进程，破坏死锁的必要条件，环路，从而恢复正常
  ③合理仔细分配资源，动态避免死锁，使用银行家算法，对于进程的每一个资源请求，都会进行检查分配资源后是否会处于安全状态，如果这一请求能保证安全状态，就分配资源，否则，推迟分配。
  ④ 破坏产生死锁的 4 个必要条件，破坏其中一个，就可以破坏死锁，防止死锁产生。

5. 实验感想

本次实验完成银行家算法，总的来说，再一次很好地复习了一遍算法，之前老师上课的时候，有点开小差，没怎么听懂，通过这一次将银行家算法用 c++写出来，终于弄清楚了。但是对于死锁问题还是有些弄不明白，银行家算法能确保不出现死锁，但是银行家算法不实用，所以解决死锁问题，我觉得还是比较困难的，对于如何能更好地解决死锁问题，不是很懂。