操作系统课程实验报告

学生姓名：/

班 学 号：/

指导教师：/

地理与信息工程学院

2019 年 5 月 17 日

实习题目：银行家算法的设计与实现

【需求规格说明】

银行家算法的设计与实现

对 I/O 系统中死锁问题求解的主要方法是银行家算法，单种资源的银行家算法和多种资源的银行家算法的解决思路一致，要求设计实现多种银行家算法，并要求所涉及的模型最少能够满足如下要求：

(1) 程序能够根据进程的请求进行判断，给出系统是否安全的的提示，如果安全，要求能够显示一组进程执行的安全序列；

(2) 能够根据需要，显示当前系统中各种资源的分配情况；

【算法设计】

（1） **设计思想：**

死锁的概念：死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。 此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。

安全和不安全状态: 如果所有过程有可能完成执行（终止），则一个状态（如上述范例）被认为是安全的。由于系统无法知道什么时候一个过程将终止，或者之后它需要多少资源，系统假定所有进程将最终试图获取其声明的最大资源并在不久之后终止。在大多数情况下，这是一个合理的假设，因为系统不是特别关注每个进程运行了多久（至少不是从避免死锁的角度）。此外，如果一个进程终止前没有获取其它能获取的最多的资源，它只是让系统更容易处理。

我们的任务就是利用银行家算法避免死锁的发生，基于这一假设，该算法通过尝试寻找允许每个进程获得的最大资源并结束（把资源返还给系统）的进程请求的一个理想集合，来决定一个状态是否是安全的。不存在这个集合的状态都是不安全的。如果申请资源后处于不安全状态就拒绝执行，安全则返回一个安全序列。

（2） **详细设计：**

我主要设计了一个类 Bank，来模拟多种资源银行家算法：

成员变量说明:

int process;	纪录进程数
int resource;	纪录资源数
*int *available;*	可利用资源数向量
*int **allocation;*	当前分配矩阵
*int **max;*	最大需求矩阵
*int **need;*	需求矩阵(max-allocation)

*int *work;*	工作向量，纪录当前可利用资源数量
*bool *finish;*	纪录进程是否完成的向量
*int **allocation_copy;*	allocation 的拷贝，便于还原
*int **workRecord;*	纪录每次的 work，便于打印信息
*int *sequence;*	一个安全序列纪录，便于输出安全序列
int rProcess;	当前请求的进程项
*int *rResource;*	当前进程请求的资源向量

函数说明：

Bank();	构造函数
~Bank();	析构函数
void init();	初始化函数
void print();	打印当前系统信息
void printWork();	打印执行过程
bool request();	发起资源请求
void reduction();	还原
bool checkSecurity();	安全性检测（主要）
bool checkOneProcess(int i);	检测一个进程

程序执行过程：

*Bank()*构造函数调用 *init()* 从键盘获取输入初始化各成员变量申请相关内存 -> 打印当前信息 -> 检测没有请求时这时是否是处于安全状态，若是输出一个安全序列；

然后选择执行 *request()*；请求资源，输入请求资源的进程以及对各资源的请求数量，检测输入，其中进程必须是存在的进程，资源请求数量必须小于等于 *max-allocation*；若输入合法，则：*need-=request；allocation+=request；work-=request；* 之后再进行安全检测；-> 若安全输出一个安全序列；

bool checkSecurity()函数（重点）：

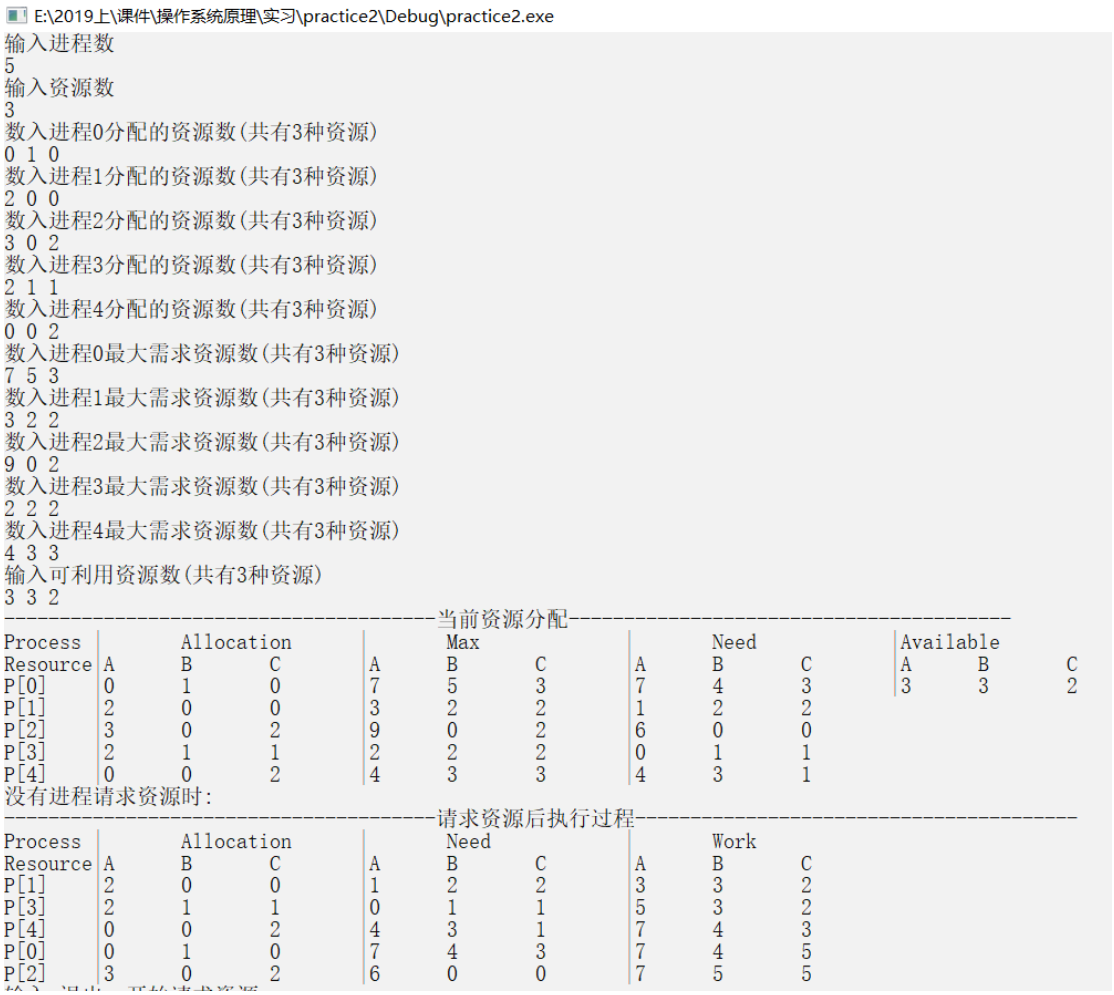
其实这个算法很简单，就是循环找出 *need<work* 的进程，更新相关变量，若最后所以进程都可以执行，则处于安全状态，否则不安全。

【调试报告】

利用老师上课 ppt 上给出的例子进行测试：输入为：

已分配的资源				最大需求量			其他
	A	B	C	A	B	C	进程数：5
P0	0	1	0	7	5	3	资源数：3
P1	2	0	0	3	2	2	剩余资源：3 3 2
P2	3	0	2	9	0	2	P1 申请：（1，0，2）
P3	2	1	1	2	2	2	P4 申请：（3，3，0）
P4	0	0	2	4	3	3	P0 申请：（0，2，0）

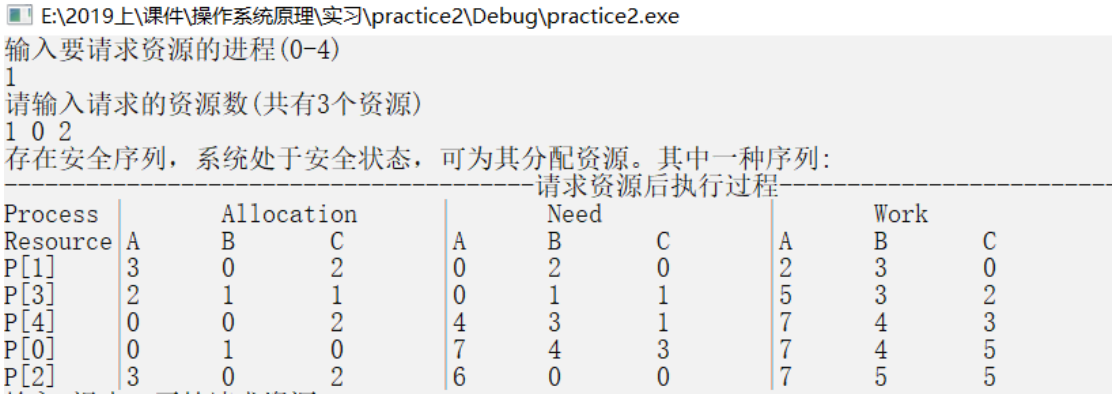
输入相关信息后，系统处于安全状态，输出一个安全序列：



安全序列为 1 -> 3 -> 4 -> 0 -> 2;

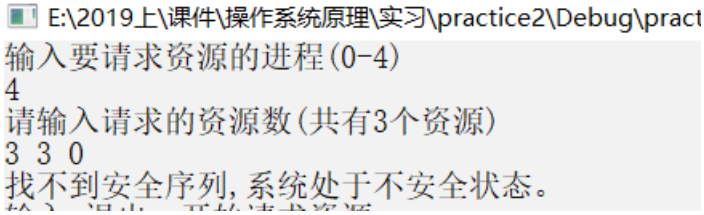
动态申请资源：

P1 申请：（1，0，2）：



申请成功，存在一个安全序列为：1 -> 3 -> 4 -> 0 -> 2;

P4 申请：（3，3，0）：



申请失败，系统处于不安全状态；

P0 申请：（0，2，0）：

```
E:\2019上\课件\操作系统原理\实习\practice2\Debug\practice2.exe
找不到安全序列,系统处于不安全状态。
输入q退出,r开始请求资源
r
输入要请求资源的进程(0-4)
0
请输入请求的资源数(共有3个资源)
0 2 0
存在安全序列，系统处于安全状态，可为其分配资源。其中一种序列:
-----------------------------------请求资源后执行过程---------------------------------
Process |       Allocation      |       Need        |       Work
Resource|A      B       C       |A      B       C   |A      B       C
P[3]    |2      1       1       |0      1       1   |3      1       2
P[1]    |2      0       0       |1      2       2   |5      2       3
P[2]    |3      0       2       |6      0       0   |7      2       3
P[0]    |0      3       0       |7      2       3   |10     2       5
P[4]    |0      0       2       |4      3       1   |10     5       5
输入q退出,r开始请求资源
```

申请成功，存在一个安全序列为：3 -> 1 -> 2 -> 0 -> 4；

【附录】

检测安全状态函数：

```
bool Bank::checkSecurity()
{
    bool deadlock = false;
    int index = 0;
    memset(finish, false, process);
    while (!deadlock&&index<process)
    {
        deadlock = true;
        for (size_t i = 0; i < process; i++)
        {
            if (!finish[i]&&checkOneProcess(i))
            {
                deadlock = false;
                sequence[index++] = i;
                finish[i] = true;
                for (size_t j = 0; j < resource; j++)
                {
                    workRecord[i][j] = work[j];
                    work[j] += allocation[i][j];
                }
            }
        }
        if (deadlock)//发生死锁
        {
            return false;
        }
    }
```

```
    return true;
}
bool Bank::checkOneProcess(int i)
{
    for (size_t j = 0; j < resource; j++)
    {
        if (work[j]<need[i][j])
        {
            return false;
        }
    }
    return true;
}
```