《银行家算法与避免死锁》实验要求

一、实验目的

1. 掌握死锁产生的必要条件及银行家算法的核心原理。

2. 通过编程实现银行家算法，模拟资源分配与安全性检查流程。

二、实验任务

1. 模拟一个银行家算法：设置数据结构设计安全性算法

2. 初始化时让系统拥有一定的资源

3. 用键盘输入的方式申请资源

4. 如果预分配后，系统处于安全状态，则修改系统的资源分配情况

5. 如果预分配后，系统处于不安全状态，则提示不能满足请求

三、实验任务要点说明

数据结构

可利用资源向量 int Available[m] m 为资源种类

最大需求矩阵 int Max[n][m] n 为进程的数量

分配矩阵 int Allocation[n][m]

还需资源矩阵 int need[i][j]=Max[i][j]-Allocation[i][j]

申请资源数量 int Request [m]

工作向量 int Work[m] int Finish[n]

银行家算法 bank()函数

Requesti：进程 Pi 的请求向量。 0<=j<=m-1

1. 若 Requesti[j]≤Need[i,j]，转向 2，否则出错。

2. 若 Requesti[j]≤Available[j]，转向 3，否则等待。

3. 系统试探着把资源分配给进程 Pi，修改下面内容：

Available[j]=Available[j]－Requesti[j];

Allocation[i,j]=Allocation[i,j]+Requesti[j];

Need[i,j]=Need[i,j]－Requesti[j];

4.试分配后，执行安全性算法，检查此次分配后系统是否处于安全状态。若安全，才正式分配；否则，此次试探性分配作废，进程 Pi 等待。

安全性算法 safe()函数

1.初始化：设置两个向量 Work(1×m)和 Finish(1×n)

Work——系统可提供给进程继续运行所需各类资源数，初态赋值 Available

Finish——系统是否有足够资源分配给进程，初值 false.

2.从进程集合中满足下面条件进程：

Finish[i] = false; Need[i,j] ≤ Work[j];

若找到，执行 3，否则，执行 4。

3.进程 Pi 获得资源，可顺利执行，完成释放所分配的资源。

Work[j] = Work[j]+Allocation[i,j];

Finish[i] = true; go to 2.

4.若所有进程 Finish[i] = true，表示系统处于安全状态，否则处于不安全状态。

先对用户提出的请求进行合法性检查，即检查请求的是否不大于需要的，是否不大于可利用的。若请求合法，则进行试分配。最后对试分配后的状态调用安全性检查算法进行安全性检查。 若安全，则分配，否则，不分配，恢复原来状态，拒绝申请。

银行家算法实例

假定系统中有五个进程{P0、P1、P2、P3、P4}和三种类型资源{A、B、C}，每一种资源的数量分别为 10、5、7。各进程的最大需求、T0 时刻资源分配情况如下所示。

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3			
P1	3	2	2	2	0	0	1	2	2			
P2	9	0	2	3	0	2	6	0	0	3	3	2
P3	2	2	2	2	1	1	0	1	1			
P4	4	3	3	0	0	2	4	3	1			

问：

①T0 时刻是否安全？

② T0 之后的 T1 时刻 P1 请求资源 Request1(1,0,2)是否允许？

③ T1 之后的 T2 时刻 P4 请求资源 Request4(3,3,0)是否允许？

④ T2 之后的 T3 时刻 P0 请求资源 Request0(0,2,0)是否允许？

四、实验报告注意事项

包含算法设计思路（流程图、代码等）。

测试用例的运行结果与分析（截图或日志）。

编程语言：C/C++/Python/Java（任选其一）。

最后需要撰写实验总结与分析（实验总结：比如讨论银行家算法的局限性（如静态进程数、固定资源类型等）……）。

***每次回收一个进程的资源后，必须从头开始重新检测所有进程。**