《计算机操作系统》实验报告

实验题目：死锁观察与避免

姓名：严昕宇　　学号：20121802　　实验日期：2022.12.09

实验环境：

实验设备：Lenovo Thinkbook16+ 2022

开发环境：CLion 2022.3

实验目的：

死锁会引起计算机工作僵死，造成整个系统瘫痪。因此，死锁现象是操作系统特别是大型系统中必须设法防止的。学生应独立的使用高级语言编写和调试一个系统动态分配资源的简单模拟程序，观察死锁产生的条件，并采用适当的算法，有效的防止死锁的发生。通过实习，更直观地了解死锁的起因，初步掌握防止死锁的简单方法，加深理解课堂上讲授过的知识。

实验要求：

1. 设计一个 n 个并发进程共享 m 个系统资源的系统。进程可动态地申请资源和释放资源。系统按各进程的申请动态地分配资源

2. 系统应能显示各进程申请和释放资源以及系统动态分配资源的过程，便于用户观察和分析

3. 系统应能选择是否采用防止死锁算法或选用何种防止算法(如有多种算法)。在不采用防止算法时观察死锁现象的发生过程。在使用防止死锁算法时，了解在同样申请条件下，防止死锁的过程

实验内容：

1. 题目

本示例采用银行算法防止死锁的发生。假设有三个并发进程共享十个系统。在三个进程申请的系统资源之和不超过 10 时，当然不可能发生死锁，因为各个进程申请的资源都能满足。在有一个进程申请的系统资源数超过 10 时，必然会发生死锁。应该排除这二种情况。程序采用人工输入各进程的申请资源序列。

如果随机给各进程分配资源，就可能发生死锁，这也就是不采用防止死锁算法的情况。假如，按照一定的规则，为各进程分配资源，就可以防止死锁的发生。示例中采用了银行算法。这是一种犹如“瞎子爬山”的方法，即探索一步，前进一步，行不通，再往其他方向试探，直至爬上山顶。这种方法是比较保守的。所花的代价也不小。

2. 算法与框图

银行算法，顾名思义是来源于银行的借贷业务，一定数量的本金要应付各种客户的借贷周转，为了防止银行因资金无法周转而倒闭，对每一笔贷款，必须考察其最后是否能归还。研究死锁现象时就碰到类似的问题，有限资源为多个进程共享，分配不好就会发生每个进程都无法继续下去的死锁僵局。

银行算法的原理是先假定每一次分配成立，然后检查由于这次分配是否会引起死锁，即剩下的资源是不是能满足任一进程完成的需要。如这次分配是安全的(不会引起死锁)，就实施这次分配，再假定下一次分配。如果不安全，就不实施，再作另一种分配试探，一直探索到各进程均满足各自的资源要求，防止了死锁的发生。

程序框图如**图 2、3、4**。

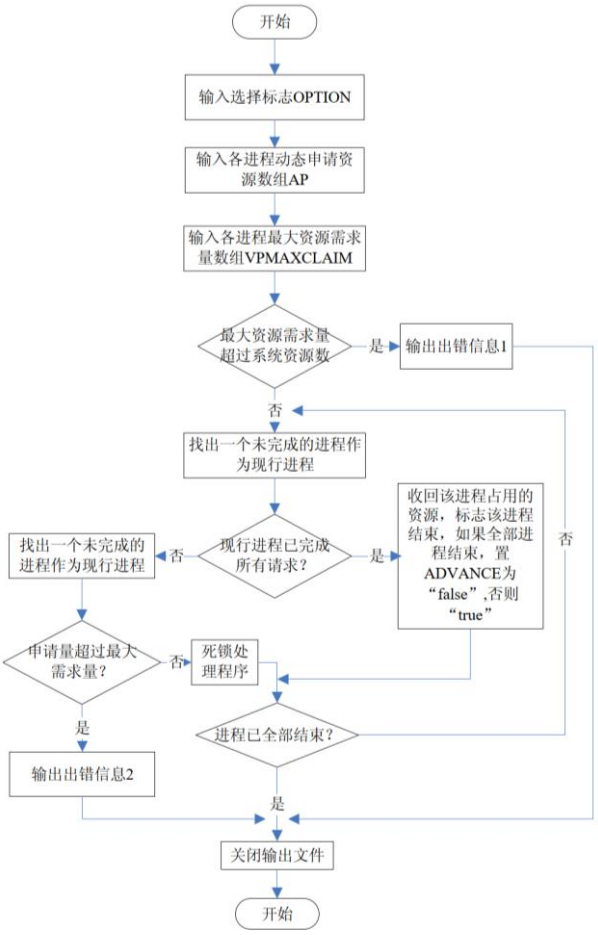


图 2 防止死锁程序框图

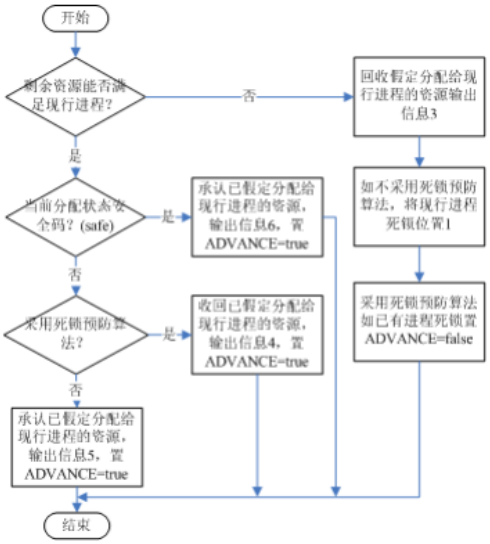


图 3 死锁处理程序框图

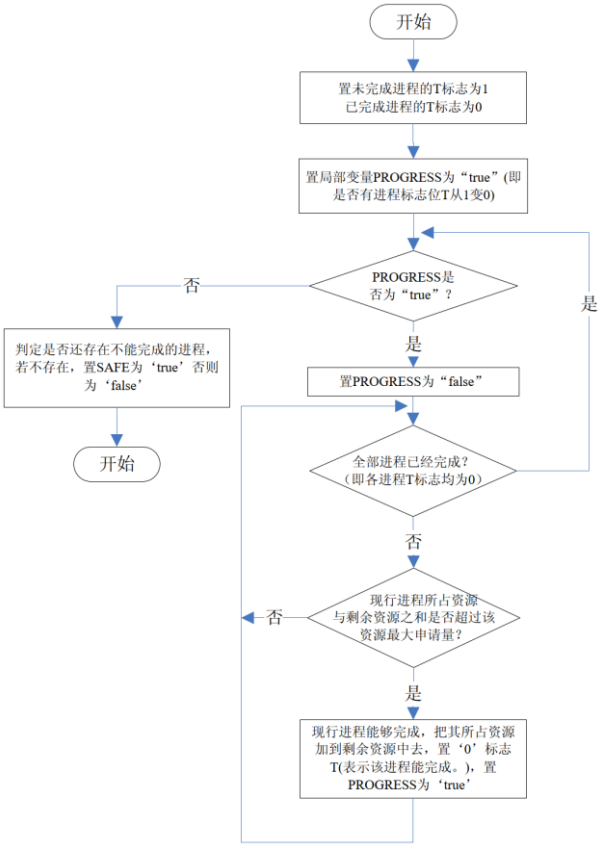


图4 safe函数框图

3. 程序运行结果格式

(1) 程序运行结果格式

```
INPUT:
OPTION =0
CLAIM OF PROCESS 1 IS: 1    2    3    -1    -1    0
CLAIM OF PROCESS 2 IS: 2    3    1    1     -2    0
CLAIM OF PROCESS 3 IS: 1    2    5    -1    -2    0
MAXCLAIM OF PROCESS 1 IS: 6
MAXCLAIM OF PROCESS 2 IS: 7
MAXCLAIM OF PROCESS 3 IS:8
THE SYSTEM ALLOCTION PROCESS IS AS FOLLOWS:
                PROCESS      CLAIM      ALLOCATION      REMAINDER
(1)             1            1          1               9
RESOURCE IS ALLOCATED TO PROCESS 1
(2)             2            2          2               7
RESOURCE IS ALLOCATED TO PROCESS 2
(3)             3            1          1               6
RESOURCE IS ALLOCATED TO PROCESS 3
(4)             1            2          3               4
RESOURCE IS ALLOCATED TO PROCESS 1
(5)             2            3          2               4
IF ALLOCATED,DEADLOCK MAY OCCUR
(6)             1            2          3               4
THE REMAINDER IS LESS THAN PROCESS 2 CLAIMS
(7)             3            0          0               10
PROCESS 3 HAS FINISHED, RETURN ITS RESOURSE
THE WHOLE WORK IS COMPLETED
********************************************
```

(2) 程序中使用的数据结构和变量名说明如下：

OPTION	选择标志
	=0 选用“防止死锁”算法
	=1 不用“防止死锁”算法

AP(I,J)　　资源请求矢量

$$= \begin{cases} n>0 & \text{第 I 进程第 J 次申请 n 个资源；} \\ n<0 & \text{第 I 进程第 J 次释放 n 个资源；} \\ n=0 & \text{第 I 进程在第 J 次终止。} \end{cases}$$

VPMAXCLAIM(I)　第 I 进程对资源的最大需求量；
VALLOCATION(I)　第 I 进程已分配到资源数；
VPSTATUS(I)　第 I 进程完成请求标志，为 1 时表示已完成各次请求；
VCOUNT(I)　第 I 进程请求次数计数器。其值表示该进程的第几次请求；
TOTAL　已分配的系统资源总数；
REMAINDER　剩余的系统资源数；
INQUIRY　当前运行进程号。

(3) 程序中定义的过程和函数说明如下：

FRONT 过程　初始化过程，装入所有初始数据，为各有关变量置初值，检查每个进程请求的资源总数是否超过系统所能提供的资源数。
PRINT 过程　输出一次分配结果。
RETRIEVE 过程　当测得资源不够分配或分配后可能产生死锁，回收已假定分配了的资源。
TERMINATION 过程　检查每个进程时候都已完成或者发生死锁。如果进程全部完成或发生死锁，则将全局变量 ADVANCE 置成 true，不然置成 false。
R_2 过程　为检查进程的分配资源数是否超过了它的最大申请量，或是释放的资源数是否超过占有数，这里是检查例外情况。

R_4 过程　为各进程设置能执行完标志 T，对于还不能完成的进程将它的标志 T 置 1，能完成的进程标志 T 置 0.
SAFE 函数　测试在当前分配状态下，会不会产生死锁，若不会，死锁函数值返回 true，否则返回 false。
ALLOCATE 过程　按申请想当前进程分配或收回资源。
RETURN 过程　收回当前进程的全部资源，并将此进程的 VPSTATUS 置 1。

(4) 程序启动后要求输入各进程的资源请求序列和各种进程的最大申请资源数。同时，要求输入选择标志 OPTION。程序运行后，输出相应结果和有关信息。输出格式如下(详见运行结果输出)：

（I） n_1　n_2　n_3　n_4
其中：
I　自然序号
n_1（PROCESS）　进程号
n_2（CLAIM）　本次申请资源数
n_3（ALLOCATION）　已分配给该进程的资源数
n_4（REMAINDER）　系统的总剩余资源数

4. 小结

本实习简单地模拟了进程调度的二种方法，在进程运行时间和进程状态变化方面都做了简化，但已能反映进程调度的实质。通过实习能加深对进程调度的理解和熟悉它的实施方法。

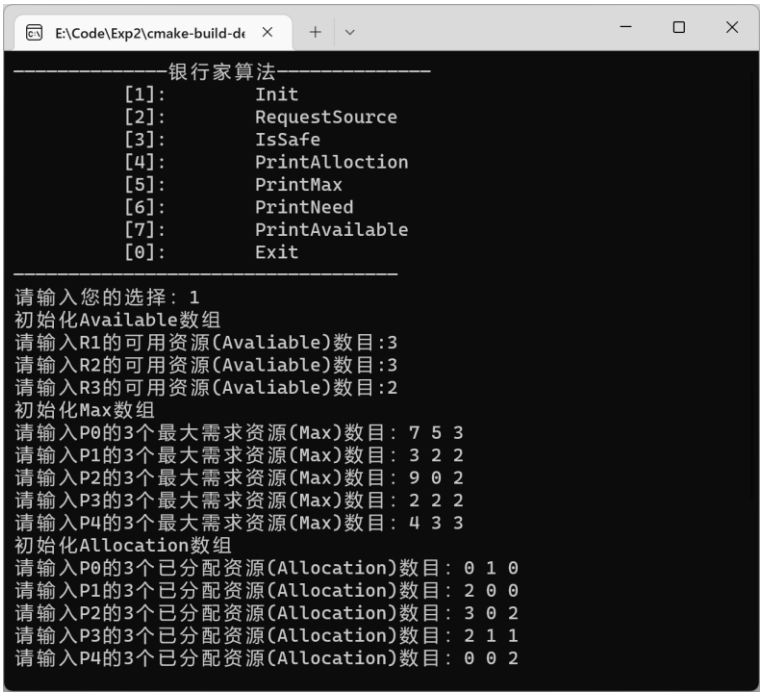
结果：

此处以 PPT 上的例题作为数据来源，对程序正确性进行检查。

【题目】某时刻系统资源分配情况如下图所示：

资源 / 进程	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	3	3	2
P1	3	2	2	2	0	0	1	2	2			
P2	9	0	2	3	0	2	6	0	0			
P3	2	2	2	2	1	1	0	1	1	资源总数		
P4	4	3	3	0	0	2	4	3	1	10	5	7

将数据输入银行家算法程序：



(1) 该时刻 T0 系统是安全的吗？

解：利用安全性算法对该时刻的资源分配情况进行分析，方法如下图：

资源 / 进程	Work			Need			Allocation			Work＋Allocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P1	3	3	2	1	2	2	2	0	0	5	3	2	true
P3	5	3	2	0	1	1	2	1	1	7	4	3	true
P4	7	4	3	4	3	1	0	0	2	7	4	5	true
P2	7	4	5	6	0	0	3	0	2	10	4	7	true
P0	10	4	7	7	4	3	0	1	0	10	5	7	true

安全序列

使用银行家算法程序，可得：

(2) 若此时 P1 请求资源，发出请求向量 Request1 (1,0,2) 系统可以满足请求吗？

解：系统按银行家算法进行检查：

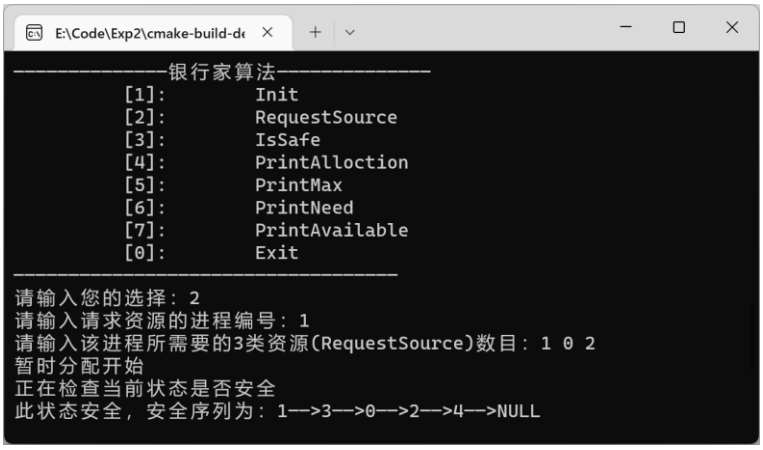
- Request1 (1,0,2)<=Need1(1,2,2)
- Request1 (1,0,2)<=Available(3,3,2)
- 系统先假定可为 P1 分配资源，修改相关向量值：
- 利用安全性算法检查此时系统是否安全。

具体：

资源 / 进程	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	2	3	0
P1	3	2	2	3	0	2	0	2	0			
P2	9	0	2	3	0	2	6	0	0			
P3	2	2	2	2	1	1	0	1	1	资源总数		
P4	4	3	3	0	0	2	4	3	1	10	5	7

资源 / 进程	Work			Need			Allocation			Work+ Allocation			Finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P1	2	3	0	0	2	0	3	0	2	5	3	2	true
P3	5	3	2	0	1	1	2	1	1	7	4	3	true
P4	7	4	3	4	3	1	0	0	2	7	4	5	true
P0	7	4	5	7	4	3	0	1	0	7	5	5	true
P2	7	5	5	6	0	0	3	0	2	10	5	7	true

使用银行家算法程序，可得：



(3) **若此时 P4 请求资源，发出请求向量 Request4(3,3,0)系统可以满足请求吗？**

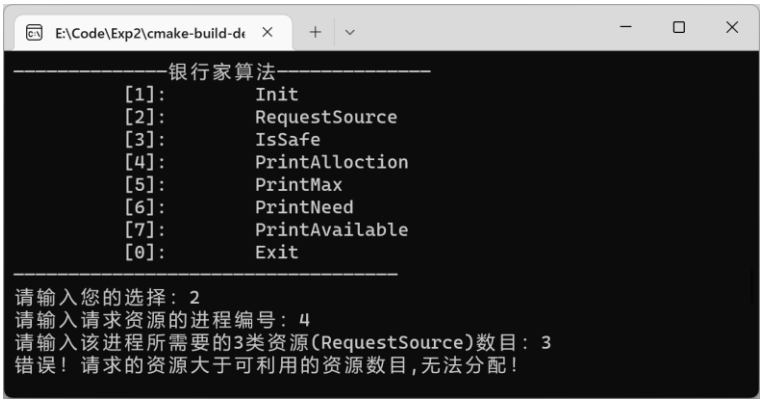
解：系统按银行家算法进行检查：

- Request4 (3,3,0)<=Need4(4,3,1)
- Request4 (3,3,0)>Available(2,3,0)

因此，让 P4 等待。

资源 / 进程	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	3	0	7	2	3	2	1	0
P1	3	2	2	3	0	2	0	2	0			
P2	9	0	2	3	0	2	6	0	0			
P3	2	2	2	2	1	1	0	1	1	资源总数		
P4	4	3	3	0	0	2	4	3	1	10	5	7

使用银行家算法程序，可得：



思考题

1．编制一个利用可在使用资源图的资源请求矩阵和分配矩阵的简化运算来检测死锁的模拟程序。

【提示】可再使用资源图的分配矩阵为(A_{ij})，元素$A_{ij}=|(R_j,P_i)|$；请求矩阵为(B_{ij})，元素$B_{ij}=|(P_i,R_j)|$，还需要一个可用资源向量(r_j)，元素$r_j=t_j-\sum_k|(R_j,P_k)|$。其中，$P$表示进程，$R$表示资源，$t$表示某类资源的总数。

【简化过程】先设法满足请求边，使它变成分配边，然后把只有分配边而无请求边的节点的所有分配边撤销(相当于解放已全部满足某进程的所有资源)。再次检查能否将请求边变成分配边，再撤销已无请求边的所有分配边，直至撤销所有的边，此时，可再使用资源图完全可化简。如果状态是死锁，当且仅当它的可再使用资源图不是完全可化简的。也就是肯定有一些边无法撤消。本实习要求用矩阵的运算来实现这过程。

答：

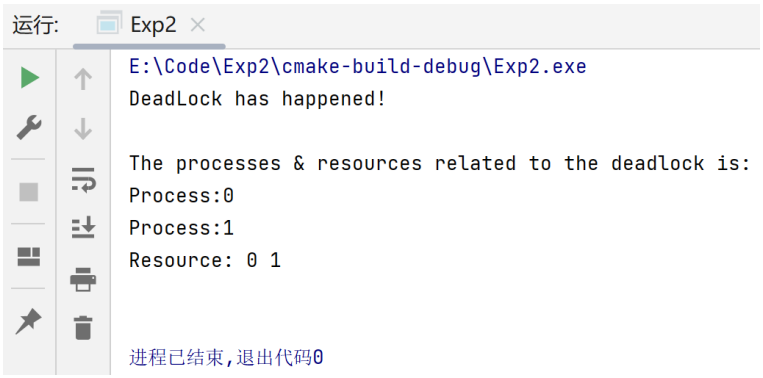
(1) 模块功能说明

- 主函数模块 main()： 在其中调用读入模块和化简模块，只是个驱动。
- 读入模块 GetData()：此模块分为两个部分(对应于文件的格式)，首先是读入资源的情况，包括资源的种类数和每类资源的数目。然后是进程的情况，包括进程的个数，每个进程占有的资源和申请的资源。所有的输入都经过合法性检查。
- 化简模块 Handle()：首先调用初始画图模块根据初始情况下的进程资源关系画出最初的资源分配图。然后，预处理那些不申请任何资源便可以运行的进程。然后进入化简的主体部分，此时，需要设置一个标志以表示这轮扫描是否化简了进程，若这轮扫描化简了进程，则应该进行下一轮扫描，因为可能这轮释放的资源可以使用前不可以化简的进程在下一轮扫描时被化简。否则，不再进行下一轮扫描。然后，把所有的未被化简的进程放到一个 Vector 中。判断这个 Vector 是否为空，不为空则表示有进程发生了死锁。输出相应的提示信息后，初始化 DFS 在中需要的一些数据结构。然后，对每一个死锁进程，若它未被访问过，则由它作为入口，进入 DFS，搜索所有和它的死锁有关系的进程和资源。当搜索结束后，输出相关的资源。若 Vector 为空，则表示所有的进程都顺利的运行完，则只要简单的输出提示信息即可。
- 输入的资源分配图文件格式：按以下次序在文件中存放数据：资源种类、每类资源的数目、进程数目、每个进程的相关信息(进程号、进程拥有的资源数目、每个资源的种类号、进程申请的资源数目、申请的资源种类号)。

(2) 程序测试：

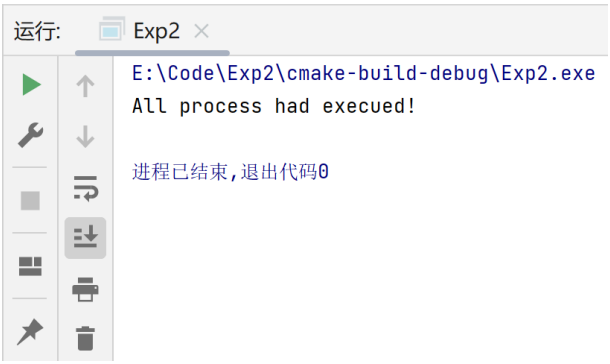
① 死锁情况：

输入文件：2 1 1 2 0 1 0 1 1 1 1 1 1 0



② 不死锁情况：

输入文件：2 3 2 2 0 2 0 0 1 1 1 2 0 1 1 0



实验体会

本次实验中，基于 C++，我实现了银行家算法，并对操作系统(1)中所讲授的系统资源管理和分配有了更深刻的了解。从该实验中，我了解到了银行家算法是个复杂度较高的算法，因此，死锁的防止是比较复杂的。虽然其可用于防死锁，但花费的代价是很大的。

通过查阅资料，我也发现，目前现代的大多数操作系统，包括 Unix、Linux 和 Windows，处理死锁问题的办法仅仅是忽略它，即鸵鸟策略。其正是因为解决死锁问题的代价很高，因鸵鸟策略这种不采取任务措施的方案会获得更高的性能。当发生死锁时不会对用户造成多大影响，或发生死锁的概率很低，可以采用鸵鸟策略。