Project 6: Banker's Algorithm

课本中提供了两种可供选择的语言——C 或者 java。我选择使用 C 语言来完成该项目。

（一）问题分析

需要实现银行家算法。

为了实现银行家算法，需要有几个数据结构。这些数据结构对资源分配系统的状态进行了记录。我们需要以下数据结构，这里 n 为系统进程的数量，m 为资源类型的种类：

- Available：长度为 m 的向量，表示每种资源的可用实例数量。如果 Available[j] = k，那么资源类型 R_j 有 k 个可用实例。
- Max：$n \times m$ 矩阵，定义每个进程的最大需求。如果 Max[i][j] = k，那么进程 P_i 最多可申请资源类型 R_j 的 k 个实例。
- Allocation：$n \times m$ 矩阵，定义每个进程现在分配的每种资源类型的实例数量。如果 Allocation[i][j] = k，那么进程 P_i 现在已分配了资源类型 R_j 的 k 个实例。
- Need：$n \times m$ 矩阵，表示每个进程还需要的剩余资源。如果 Need[i][j] = k，那么进程 P_i 还可能申请 k 个资源类型 R_j 的实例。注意 Need[i][j] = Max[i][j] − Allocation[i][j]。

这些数据结构的大小和值会随着时间而改变。

为了简化银行家算法的描述，下面采用一些符号。设 X 和 Y 为长度为 n 的向量。我们说：$X \leqslant Y$ 当且仅当对所有 $i = 1, 2, \cdots, n$，$X[i] \leqslant Y[i]$。例如，如果 $X = (1, 7, 3, 2)$ 而 $Y = (0, 3, 2, 1)$，那么 $Y \leqslant X$。此外，如果 $Y \leqslant X$ 且 $Y \neq X$，那么 $Y < X$。

可以将矩阵 Allocation 和 Need 的每行作为向量，并分别用 Allocation_i 和 Need_i 来表示。向量 Allocation_i 表示分配给进程 P_i 的资源；向量 Need_i 表示进程为完成任务可能仍然需要申请的额外资源。

7.5.3.1 安全算法

现在我们介绍这个算法，以求出系统是否处于安全状态。该算法可以描述如下：

1. 令 Work 和 Finish 分别为长度 m 和 n 的向量。对于 $i = 0, 1, \cdots, n-1$，初始化 Work = Available 和 Finish[i] = false。
2. 查找这样的 i 使其满足
   a. Finish[i] == false
   b. $\text{Need}_i \leqslant$ Work
   如果没有这样的 i 存在，那么就转到第 4 步。
3. Work = Work + Allocation_i
   Finish[i] = true
   返回到第 2 步。
4. 如果对所有 i，Finish[i] = true，那么系统处于安全状态。

这个算法可能需要 $m \times n^2$ 数量级的操作，以确定系统状态是否安全。

7.5.3.2 资源请求算法

现在，我们描述判断是否安全允许请求的算法。

设 $Request_i$ 为进程 P_i 的请求向量。如果 $Request_i[j] == k$，那么进程 P_i 需要资源类型 R_j 的实例数量为 k。当进程 P_i 作出这一资源请求时，就采取如下动作：

1. 如果 $Request_i \leqslant Need_i$，转到第 2 步。否则，生成出错条件，这是因为进程 P_i 已超过了其最大需求。

2. 如果 $Request_i \leqslant Available$，转到第 3 步。否则，$P_i$ 应等待，这是因为没有资源可用。

3. 假定系统可以分配给进程 P_i 请求的资源，并按如下方式修改状态：

$$Available = Available - Request_i$$

$$Allocation_i = Allocation_i + Request_i$$

$$Need_i = Need_i - Request_i$$

如果新的资源分配状态是安全的，那么交易完成且进程 P_i 可分配到需要的资源。然而，如果新状态不安全，那么进程 P_i 应等待 $Request_i$ 并恢复到原来的资源分配状态。

（二）实现细节

一、全局变量

1、available[]: the available amount of each resource

2、maximum[][]: the maximum demand of each customer

3、allocation[][]: the amount currently allocated to each customer

4、need[][]: the remaining need of each customer

5、request[]: 记录 RQ 指令请求的资源

6、release[]: 记录 RL 指令想要释放的资源

二、函数

1、request_resource()

（1)根据 custom_num 和 request[]计算新状态的 new_available[]、new_allocation[][]、new_need[][].计算过程中可做简单判断:如果 request > available 或者 new_allocation > maximum，那么立刻可以判断出该请求不能被批准。

（2）判断新状态是否为安全状态。用 new_available[]初始化 work[]，用 finish[]记录进程的完成状态。遍历寻找下一个可以完成的进程，即未完成且 need < work 的进程。若找到(choose == i)，则更新 finish[i]和 work[i]；若未找到(choose == -1)，则遍历检查当前是否还有未完成的进程。若有，则说明有进程被死锁，新状态为非安全状态，请求不能被批准；若没有，则说明所有进程都可以完成，新状态为安全状态，请求能被批准，所以用新状态更新当前状态。

2、release_resource()

首先根据 custom_num 和 release[]检查是否有足够的资源能被释放：如果 release > allocate，则释放指令无法执行。如果不存在以上情况，则更新 available[]、allocation[][]、need[][]。

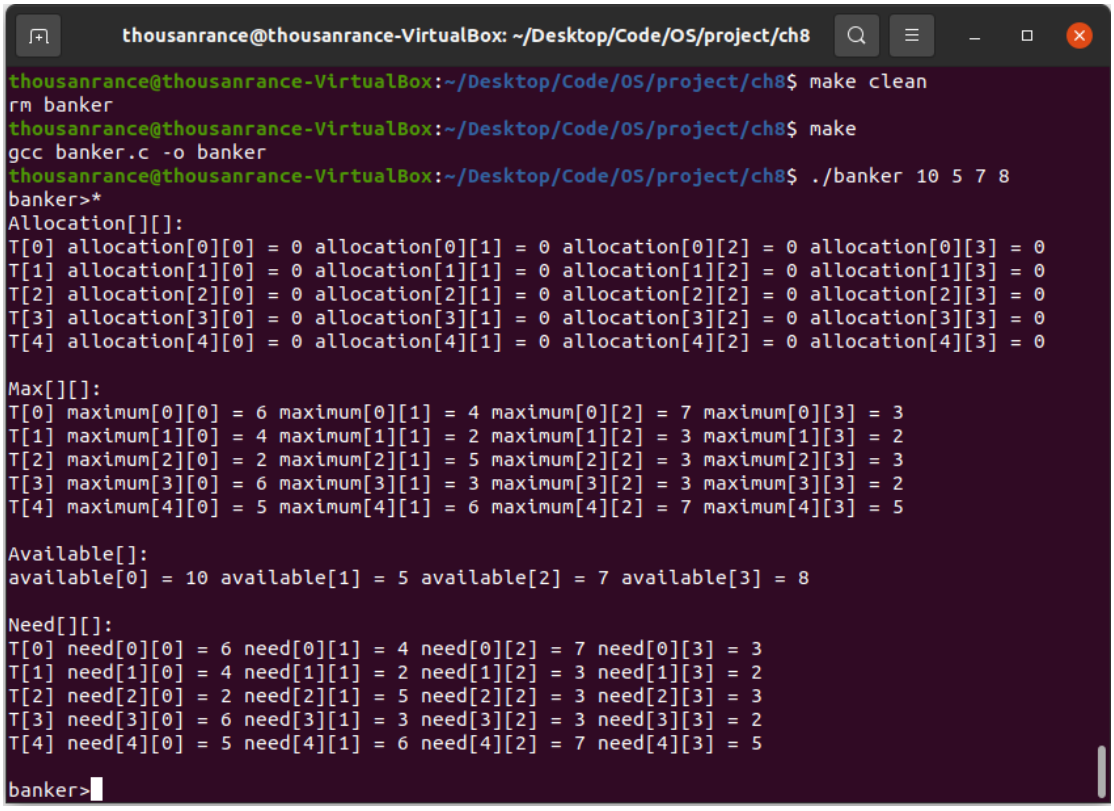
3、show_current_state()

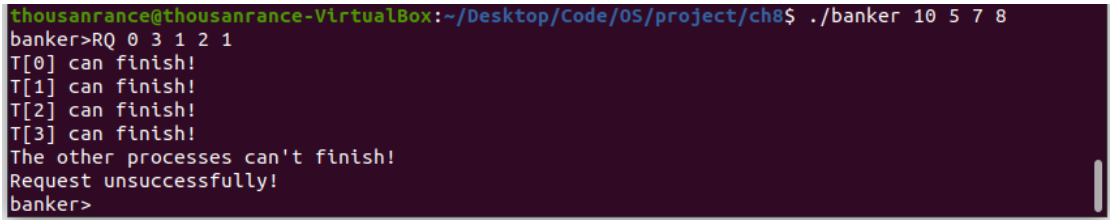
打印当前的 allocation[][]、maximum[][]、available[]、need[][]。

三、主函数

1、从命令行参数获取 available[]
2、从 maximum.txt 文件读取 maximum[][]
3、初始化 allocation[][]和 need[]
4、仿照 project2-1：Unix Shell 写命令行“banker>”主循环代码框架。
5、input[]存储输入的指令。
6、RQ 指令
处理 input[]得到 custom_num 和 request[]，调用函数 request_resources()，根据返回值输出信息。
7、RL 指令
处理 input[]得到 custom_num 和 release[]，调用函数 release_resources()。
8、* 指令
调用函数 show_current_state()。
9、exit 指令
退出 banker 命令行，即程序终止

（三）运行结果





```
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch8$ ./banker 10 6 7 8
banker>RQ 0 3 1 2 1
T[0] can finish!
T[1] can finish!
T[2] can finish!
T[3] can finish!
T[4] can finish!
Request successfully!
banker>RL 0 1 1 1 1
banker>*
Allocation[][]:
T[0] allocation[0][0] = 2 allocation[0][1] = 0 allocation[0][2] = 1 allocation[0][3] = 0
T[1] allocation[1][0] = 0 allocation[1][1] = 0 allocation[1][2] = 0 allocation[1][3] = 0
T[2] allocation[2][0] = 0 allocation[2][1] = 0 allocation[2][2] = 0 allocation[2][3] = 0
T[3] allocation[3][0] = 0 allocation[3][1] = 0 allocation[3][2] = 0 allocation[3][3] = 0
T[4] allocation[4][0] = 0 allocation[4][1] = 0 allocation[4][2] = 0 allocation[4][3] = 0

Max[][]:
T[0] maximum[0][0] = 6 maximum[0][1] = 4 maximum[0][2] = 7 maximum[0][3] = 3
T[1] maximum[1][0] = 4 maximum[1][1] = 2 maximum[1][2] = 3 maximum[1][3] = 2
T[2] maximum[2][0] = 2 maximum[2][1] = 5 maximum[2][2] = 3 maximum[2][3] = 3
T[3] maximum[3][0] = 6 maximum[3][1] = 3 maximum[3][2] = 3 maximum[3][3] = 2
T[4] maximum[4][0] = 5 maximum[4][1] = 6 maximum[4][2] = 7 maximum[4][3] = 5

Available[]:
available[0] = 8 available[1] = 6 available[2] = 6 available[3] = 8

Need[][]:
T[0] need[0][0] = 4 need[0][1] = 4 need[0][2] = 6 need[0][3] = 3
T[1] need[1][0] = 4 need[1][1] = 2 need[1][2] = 3 need[1][3] = 2
T[2] need[2][0] = 2 need[2][1] = 5 need[2][2] = 3 need[2][3] = 3
T[3] need[3][0] = 6 need[3][1] = 3 need[3][2] = 3 need[3][3] = 2
T[4] need[4][0] = 5 need[4][1] = 6 need[4][2] = 7 need[4][3] = 5

banker>exit
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch8$
```