广州大学学生实验报告

**开课学院及实验室：**计算机科学与网络工程学院软件实验室 2019**年**11**月**7**日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | 计算机科学与网络工程学院 | **年级/专业/班** | 软件171 | **姓名** | 谢金宏 | **学号** | 1706300001 |
| **实验课程名称** | 操作系统实验 | | | | | **成绩** |  |
| **实验项目名称** | 银行家算法 | | | | | **指导老师** | 陶文正 |

实验二 银行家算法

# 实验目的

1. 了解什么是操作系统安全状态和不安全状态；
2. 了解如何避免系统死锁；
3. 理解银行家算法是一种最有代表性的避免死锁的算法，掌握其实现原理及实现过程。

# 实验环境

任意POSIX兼容的操作系统编程环境：

1. Windows Subsystem Linux (WSL)
2. CentOS

# 实验内容

根据银行家算法的基本思想，编写和调试一个实现动态资源分配的模拟程序，并能够有效避免死锁的发生。

# 实验原理

进程申请资源时，系统通过一定的算法判断本次申请是否不可能产生死锁（处于安全状态）。若可能产生死锁（处于不安全状态），则暂不进行本次资源分配，以避免死锁。算法有著名的银行家算法。

## 安全状态和不安全状态

所谓安全状态，是指如果系统中存在某种进程序列＜P1，P2，…，Pn＞，系统按该序列为每个进程分配其所需要的资源，直至最大需求，则最终能使每个进程都可顺利完成，称该进程序列＜P1，P2，…，Pn，＞为安全序列。

如果不存在这样的安全序列，则称系统处于不安全状态。

## 银行家算法

把操作系统看作是银行家，操作系统管理的资源相当于银行家管理的资金，进程向操作系统请求分配资源相当于用户向银行家贷款。

## 为保证资金的安全，银行家规定:

1. 当一个顾客对资金的最大需求量不超过银行家现有的资金时就可接纳该顾客;
2. 顾客可以分期贷款，但贷款的总数不能超过最大需求量;
3. 当银行家现有的资金不能满足顾客尚需的贷款数额时,对顾客的贷款可推迟支付,但总能使顾客在有限的时间里得到贷款;
4. 当顾客得到所需的全部资金后,一定能在有限的时间里归还所有的资金。

## 操作系统按照银行家制定的规则设计的银行家算法为：

1. 进程首次申请资源的分配：如果系统现存资源可以满足该进程的最大需求量，则按当前的申请量分配资源，否则推迟分配。
2. 进程在执行中继续申请资源的分配：若该进程已占用的资源与本次申请的资源之和不超过对资源的最大需求量，且现存资源能满足该进程尚需的最大资源量，则按当前申请量分配资源，否则推迟分配。
3. 至少一个进程能完成：在任何时刻保证至少有一个进程能得到所需的全部资源而执行到结束。

银行家算法通过动态地检测系统中资源分配情况和进程对资源的需求情况来决定如何分配资源，并能在确保系统处于安全状态时才把资源分配给申请者，从而避免系统发生死锁。

# 使用的系统调用函数

这是一个模拟程序，没有使用系统调用函数。

# 实验过程记录与结果分析

## 算法流程图

银行家算法的整体框图如下：

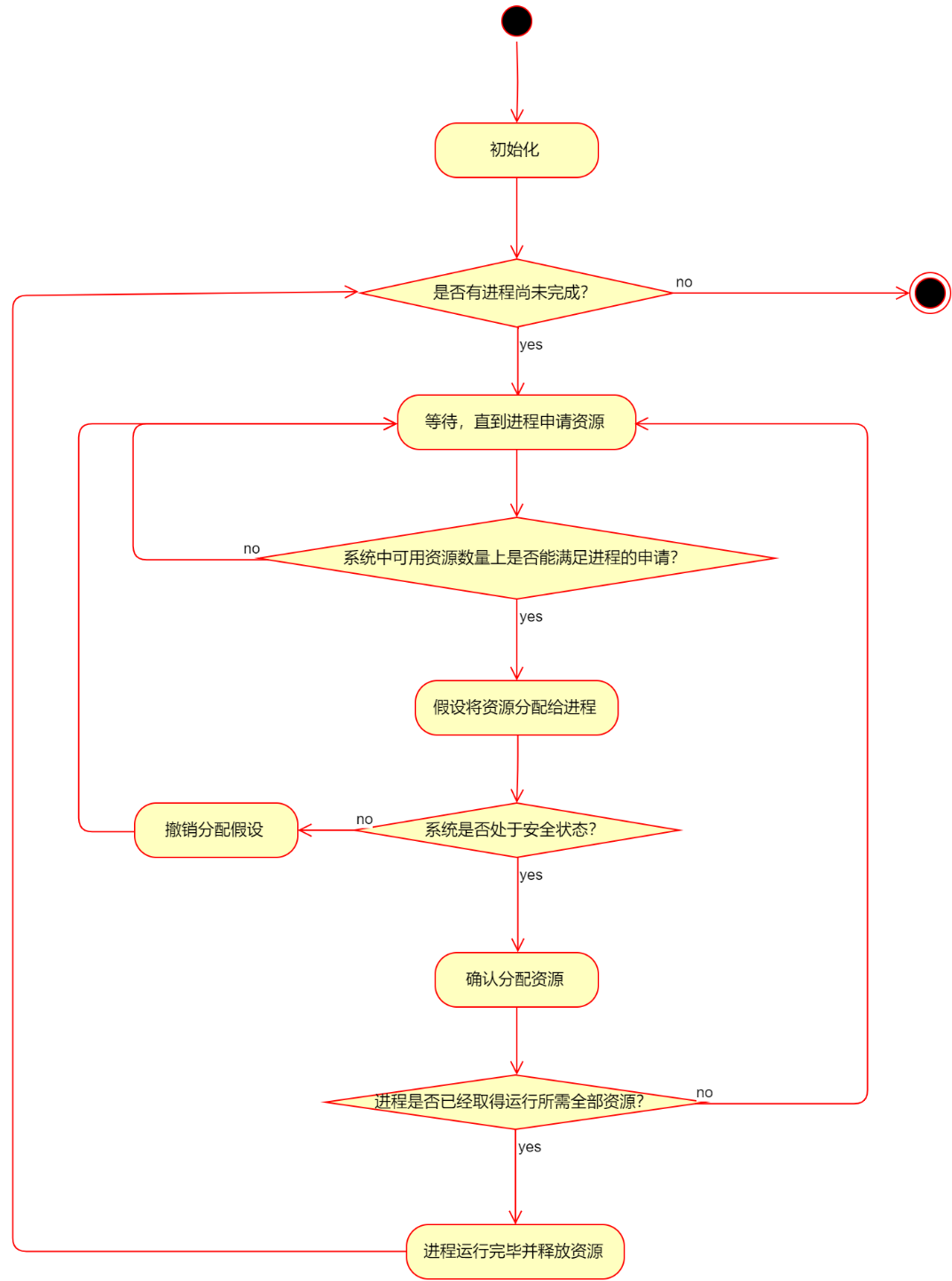


图 1 银行家算法流程图

对应流程图，程序的代码框架如下：



图 2 主要程序框架

程序开始运行时，先要进行初始化操作（init函数），随机生成系统中资源的总量、进程对资源的最大需求量，并打印这些初始值（print系列函数）。银行家算法运行时，在系统的初始状态，系统中的可用资源数量为系统资源的总量。第1步，银行家算法判断系统中是否还有进程没有完成运行（allProcessIsFinished函数）；若所有进程都已完成运行，则算法结束；若还有进程没有完成运行，则转第2步，等待进程提出资源申请（randomRequest函数）。

当某进程提出资源请求时，算法先判断系统中的可用资源数量上能否满足进程提出的请求。若数量上不能满足，则转第2步继续等待。否则，继续执行算法。先假定将系统中的资源分配给进程（tryAllocate函数），然后判断系统中是否存在一个安全序列（getSafeSequence函数）。若系统中存在安全序列，则正式将资源分配给进程（commitAllocate函数），然后转步骤1。此时，若进程已经取得运行所需的所有资源，则应开始运行，并在结束后向系统释放资源。若系统中不存在安全序列，则撤销资源分配假设（undoAllocate函数），并转步骤2继续等待。

## 数据结构说明

全局变量TOTAL\_RESOURCE为一维整数数组，TOTAL\_RESOURCE[i]保存系统中i类资源的总量。在程序初始化被赋值后不可变。

全局变量MAX\_REQUEST为二维整数数组，MAX\_REQUEST[i][j]表示进程i对资源j的最大需求量。在程序初始化被赋值后不可变。

全局变量availableResource为一维整数数组，availableResource[i]保存系统运行过程中可用分配的i类资源数量。初始值为TOTAL\_RESOURCE。

全局变量accquiredResource为二维整数数组，accquiredResource[i][j]表示进程i在运行过程中已经取得j类资源的数量。初始值为0，进程结束运行后值为0。

全局变量isFinished为一维布尔数组，isFinished[i]表示进程i是否已经结束运行。初始值为false，进程结束运行后为true。

## 使用init函数和randomRequest函数随机产生测试数据



图 3 Init函数体

Init函数负责随机生成系统中资源的总量，系统中资源的种类由常量TYPE\_OF\_RESOURCE指定，系统中资源的总量通过模运算和线性运行控制在常量上界RANDOM\_RESOURCE\_UPPER\_BOUND和常量下界RANDOM\_RESOURCE\_LOWER\_BOUND之间。Init函数还负责生成进程对资源的最大需求量。系统中进程的数量由常量NUMBER\_OF\_PROCESS指定，进程对某类资源的最大需求量通过模运算控制不超过系统中该类资源的总量。



图 4 RandomRequest函数体

RandomRequest函数负责模拟进程向操作系统提出资源申请。函数执行时，先从未完成运行的进程中挑选一个，为每类资源随机生成一些数字作为资源申请量，并通过模运算将这些数字的大小控制在MAX\_REQUEST-acquiredResource范围内，即控制在自己实际所需要的资源数量范围内。如果这些资源的申请量全为0，则要重新取随机数。

通过init函数和randomRequest函数，系统实现了随机输入。

## 使用getSafeSequence函数获取当前系统中的一个安全序列

getSaftSequence函数是银行家算法的核心。将算法的核心思想描述如下：

算法初始化一个名为安全序列的空一维整数数组。为了找出当前系统中的安全序列，算法遍历进程列表。在进程列表中选取未结束运行的进程，判断系统当前可供分配的资源数量能否满足进程的需求。若能满足进程需求，则将进程标记为结束运行状态，将进程序号加入安全序列中，并回收进程占用的所有资源，然后从列表头部重新开始遍历。若不能满足需求，则从当前位置继续遍历。当遍历至进程列表末尾时，遍历阶段结束。



图 5 银行家算法中对进程列表的遍历

结束进程遍历后，算法判断系统中是否所有进程都结束运行。如果所有进程都结束运行，那么算法得到的安全序列便是系统中的一个安全序列。若并非所有进程都能结束运行，表明系统处于不安全状态。

## 使用tryAllocate函数、commitAllocate函数和undoAllocate函数进行资源分配

tryAllocate函数与commitAllocate/UndoAllocate函数之一成对成对使用。编写资源分配函数的要点是，系统中的可用资源availableResource和进程占用的资源acquiredResource要成对地进行修改。

函数代码详见[第7大节](#_实验数据及源代码)。

## 程序运行结果记录

由于程序的输出很长，此处只截取片段展示。完整示例数据位于[实验数据](#_实验数据及源代码)中的“banker.txt”。

开始模拟时，系统的初态如下：



图 6 系统初态

模拟过程中的一些输入如下：



图 7 模拟过程中的一些输出

一段时间后，一些进程结束运行：



图 8 一些进程在一段时间后结束运行

系统终态，所有进程都运行完毕：



图 9 所有进程运行完毕

# 实验总结

总体而言完成了本次实验的全部要求。

代码实现中有可供改良的地方。本次实验中代码采用C++编写，但没有采用面向对象的方式编程。现存的问题是所有代码挤在一个编译单元中，并且大量使用了全局变量，而这些全局变量并非与所有函数都相关连。若将代码重新组织为面向对象编程方式，并将类编入相应的编译单元中，则代码可读性会有相当程度的提高。

实验中没有模拟进程提出不合法的请求的情况。所谓不合法的请求，是指进程申请资源的数量超过了它声明自己应该申请的资源的数量。

# 实验数据及源代码

实验数据及源代码见“1706300001\_谢金宏\_实验二.zip”。