

**课程设计(综合实验)报告**

( 2019 – 2020 年度第 2 学期)

名 称： 操作系统综合实验

题 目： 银行家算法

院 系： 计算机系

班 级： 软件1802

学 号： 201809000121

学生姓名： 王毅涛

指导教师： 王德文

设计周数： 分散进行

成 绩：

日期： 年 月 日

**一、 综合实验的目的与要求**

银行家算法是避免死锁的一种重要方法，要求编写和调试一个简单的银行家算法程序。有助于加深了解有关资源申请、避免死锁等概念，并体会和了解死锁和避免死锁的具体实施方法

**二、实验正文**

本实验采取JAVA语言编写，用户通过控制台输入数据。

1.实验中采用的数据结构

本实验采用客户端-主程序样式，在客户端初始化主程序后，主程序提示用户输入数值。在主程序中，含有若干数组：一维数组Available和Work,二维数组Max,Alloction,Need,Request。本次实验以三个进程，三个资源种类为例编写。初始化会将主程序的上述数组全部赋值0，然后客户端调用主程序方法，该方法调用其他方法引导用户输入数值，最后弹出结果。

2.思路：

银行家算法有专门的算法思想结束，这里不再详述，这里的MAX和ALLOCATION还有AVAILABLE通过键盘输入，然后NEED计算为（MAX-ALLOCATION）,AVAILABLE再调整。然后输出可用资源数目，各个进程的MAX,ALLOCATION,NEED数目。之后调用另一个方法引导用户输入请求资源的进程以及各个资源请求数目存入Request。接下来检查每个资源的请求数目有没有超过该进程所需的数目（Need）,TG如果超出了就提示失败并返回，未超出进入下一步；检查每个资源的请求数目有没有超过对应资源的可用数目，如果超出就提示失败返回，如果没有超出就进行到安全性检查；安全性检查是银行家算法的重要步序，在安全性检查之前先对于资源做出预分配，即把Request数值加到Allocation,Need和Available减去Request.这里的分配是预分配，如果后期安全性检查不通过是要还原的。所以这里额外设定了三个全局变量，用来存放上述三个二维数组。如果需要使用就还原。

安全性检查需要

Ⅰ设置两个工作向量 WORK=AVAILABLE；FINISH[M]=FALSE

Ⅱ从进程集合中找到一个满足下述条件的进程，

FINISH全为FALSE

同一进程里NEED中所有值都大于WORK中对应值

如找到，执行Ⅲ；否则，执行Ⅳ

Ⅲ设进程获得资源，可执行，直至完成后释放资源。

WORK=WORK+ALLOCATION

FINISH=TRUE

跳转到 Ⅱ（操作中是用BREAK跳出循环）

Ⅳ Finish中各项全为TRUE，则表示安全；否则系统不安全。

然后提示有没有完成，没有完成的话哪一步出了问题，出了什么问题

**三、综合实验总结或结论**

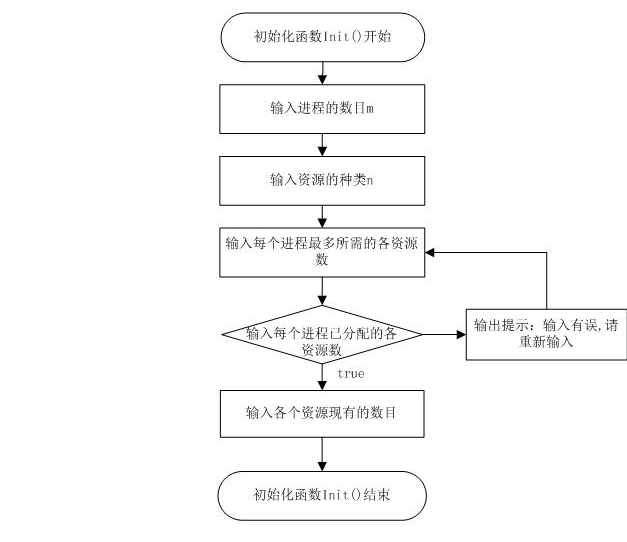
本次实验的目的基本达成，用户友好度也比上次的实验高了不少。这次实验的难点在于各种复杂的条件判断和转移，由于实验指导书上给的提示相当简略，我在网络上进行了一些资料查找，终于搞懂了银行家算法的安全性检查机制。本次实验将资源种类数目限制为3，允许进程数目为3，是出于调试方便，如果在实际的系统里，必然会有更多的需要给定的参数。

**四、参考文献**

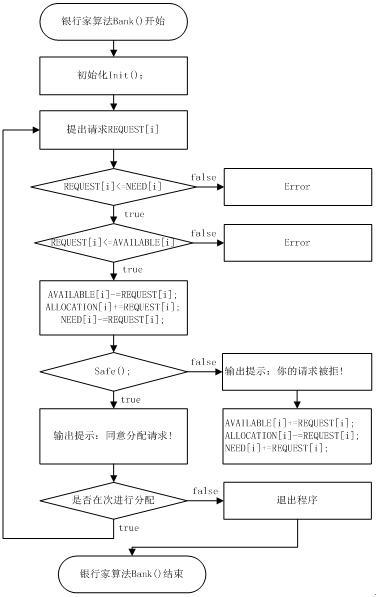
[1] 汤小丹, 梁红兵.《计算机操作系统》. 西安电子科技大学出版社, 第四版. 2014-05

**附录（设计流程图、程序、表格、数据等）**

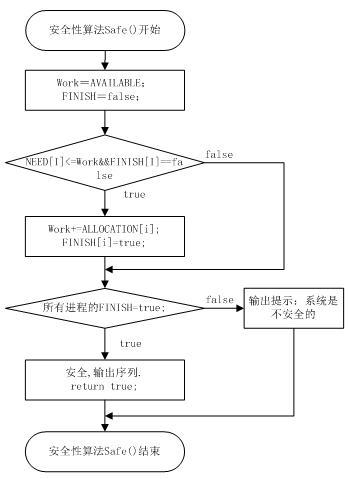
初始化（部分功能没有开放）：



预分配阶段：



安全性计算：



代码：

1. **import** java.util.List;
2. **import** java.util.Scanner;
4. //本实验数组一律从0开始
5. **public** **class** Conductor {
6. **int**[] Available ;
7. **int**[][] Max ;
8. **int**[][] Alloction ;
9. **int**[][] Need  ;
10. **int**[][] Request;
11. **int**[] Security;
12. **int** [] Work ;
13. Scanner in = **new** Scanner(System.in);
14. **public** Conductor() {
15. **super**();
16. Available = **new** **int** [3];
17. Max = **new** **int**[3][3];
18. Alloction = **new** **int**[3][3];
19. Need = **new** **int**[3][3];
20. Request = **new** **int**[3][3];
21. Security = **new** **int**[3];
22. Work =**new** **int** [3];
23. }
24. **public** **void** execute() {
26. }
27. **public** **void** initialize() {
28. **this**.setAvail();
29. **this**.setMax();
30. **this**.setAllo();
31. **this**.printme();
32. **this**.issueRequest();
33. }
35. **public** **void** setAvail() {
36. System.out.println("总可用资源指定开始，三个数值用空格或者回车分割");
37. **for** (**int** i=0;i<3;i++) {
38. **this**.Available[i]=in.nextInt();
39. }
40. System.out.println("总可用资源指定完成");
41. }
42. **public** **void** setMax() {
43. System.out.println("设定最大需求矩阵");
44. **for** (**int** i=0;i<3;i++) {
45. System.out.println("下面指定"+i+"号任务的最大资源需求,空格或者回车分开");
46. **for** (**int** j=0;j<3;j++) {
47. **this**.Max[i][j]=in.nextInt();
48. }
49. }
50. System.out.println("指定完成");
51. }
52. **public** **void** setAllo() {//这里需要更新Need和Available
53. System.out.println("开始指定已分配矩阵");
54. **for** (**int** i = 0; i < 3; i++) {
55. System.out.println("输入进程P" + i + "的分配数值");
56. **for** (**int** j = 0; j < 3; j++) {
57. Alloction[i][j] = in.nextInt();
58. }
59. }
60. System.out.println("Available=Available-Alloction.");
61. System.out.println("Need=Max-Alloction.");
62. **for** (**int** i = 0; i < 3; i++) {//Avail更新
63. **for** (**int** j = 0; j < 3; j++) {
64. Available[i] = Available[i] - Alloction[j][i];
65. }
66. }
67. **for** (**int** i = 0; i < 3; i++) {//Need指定
68. **for** (**int** j = 0; j < 3; j++) {
69. Need[i][j] = Max[i][j] - Alloction[i][j];
70. }
71. }
72. }

75. **public** **void** printme() {
76. **for** (**int** i=0;i<3;i++) {
77. //System.out.println("Available");
78. **if** (i==0) {
79. System.out.println("Available");
80. **for** (**int** j=0;j<3;j++) {
81. System.out.print(**this**.Available[j]+" ");
82. }
83. }
84. **else** {;}
85. System.out.println();
86. System.out.println("进程名   Max    Need   Allocation ");
87. //System.out.println();
88. System.out.print("P"+i);
89. System.out.print("#");
90. **for** (**int** j=0;j<3;j++) {
91. System.out.print(**this**.Max[i][j]+" ");
92. }
93. System.out.print("#");
94. **for** (**int** j=0;j<3;j++) {
95. System.out.print(**this**.Need[i][j]+" ");
96. }
97. System.out.print("#");
98. **for** (**int** j=0;j<3;j++) {
99. System.out.print(**this**.Alloction[i][j]+" ");
100. }
101. System.out.print("#");
102. }
103. }
105. **public** **void** issueRequest() {
106. System.out.println("键入想要新资源的进程");
107. **int** i=in.nextInt();
108. System.out.println("下面键入三个资源的需求量,空格或回车分开");
109. **for** (**int** j=0;j<3;j++) {
110. **this**.Request[i][j]=in.nextInt();
111. }
112. System.out.println("第"+i+"号进程要求取得三种资源的数目为"+**this**.Request[i][0]+""+**this**.Request[i][1]+""+**this**.Request[i][2]);
113. System.out.println("银行家算法审核开始");
114. **boolean** reach=**this**.core(i);
116. }
117. **public** **boolean** core(**int** num) {//算法核心，需要指定最近一次录入的进程编号
118. //不能单独使用！
119. **if** (**this**.Request[num][0]<=**this**.Need[num][0] && **this**.Request[num][1]<=**this**.Need[num][1] && **this**.Request[num][2]<=**this**.Need[num][2])
120. {
121. **if** (**this**.Request[num][0]<=**this**.Available[0] && **this**.Request[num][1]<=**this**.Available[1] && **this**.Request[num][2]<=**this**.Available[2])
122. {
123. //上面检查也没有超出预期需要，以及有没有超出资源局限
124. **for** (**int** i=0;i<3;i++) {
125. Available[i] -= Request[num][i];
126. Alloction[num][i] += Request[num][i];
127. Need[num][i] -= Request[num][i];
128. }
129. }
130. **else** {
131. System.out.println("进程"+num+"超出可用资源限制");
132. **return** **false**;
133. }
134. }
135. **else** {
136. System.out.println("进程"+num+"的请求超出它允许请求的资源数的最大值");
137. **return** **false**;
138. }
139. **this**.printme();
140. System.out.println("现在执行安全算法");
141. **boolean** T=**this**.security();
142. **if** (T) {
143. System.out.println("安全算法通过");
144. **return** **true**;
145. }
146. **else** {
147. System.out.println("安全算法阻止");
148. **return** **false**;
149. }
150. //有待补完
151. }
152. **public** **boolean** security() {
153. //Max.length表示有几个进程  Available.length表示几个资源种类
154. **boolean** [] finished= {**false**,**false**,**false**};
155. **for** (**int** i=0;i<**this**.Available.length;i++)
156. {
157. Work[i]=**this**.Available[i];
158. }
159. **int** pro=**this**.Max.length;
160. **int** res=**this**.Available.length;
161. **for** (**int** tmp=0;tmp<pro;tmp++) {
162. **for** (**int** i=0;i<pro;i++) {
163. **if** (finished[i]==**false**&&Need[i][0]<=Work[0]&&Need[i][1]<=Work[1]&&Need[i][2]<=Work[2])
164. {
165. **for** (**int** k=0;k<res;k++) {
166. **this**.Work[k]+=**this**.Alloction[i][k];
167. }
168. finished[i]=**true**;
169. **break**;
170. }
171. }
173. }
175. **for** (**int** i=0;i<pro;i++) {
176. **if** (finished[i]==**false**) {
177. **return** **false**;
178. }
179. **else** {;}
180. }
181. **return** **true**;
183. }

186. }

运行结果

