

**《操作系统》**

**银行家算法模拟实现**

|  |  |
| --- | --- |
|  |  |
| 学 号： |  |
| 姓 名： |  |
| 学 院： |  |
| 日 期： | 2022年 10 月 27 日 |

# 一、实验内容

（1）基于 C 语言的银行家算法的设计与实现；

（2）计算机系统资源的模拟管理和处理场景的构建（初始化操作包括系统各类资源配备情况、一组并发进程及相应资源最大需求明细，进程申请资源操作需要指定进程及其对所需各类资源的申请数量，进程释放资源操作需要指定进程及其对当前所占用各类资源的释放数量）；

（3）算法原型应能正确处理进程申请/释放资源的各种操作请求；

（4）针对银行家算法原型开展基于计算机系统资源管理的完备的测试验证。

# 二、开发环境、运行环境、测试环境

开发环境：Visual Studio 2022

运行环境：ubuntu虚拟机

测试环境：ubuntu虚拟机

# 三、实验步骤

## 一、关键数据结构

资源总量向量 Resource， m 维，表示 m 种资源的总量。

可用资源向量 Available， m 维，表示未分配的各种可用资源数量。

需求矩阵Need， nm 矩阵，表示 n 个进程对 m 类资源的最大需求。

分配矩阵 Allocation ， nm 矩阵，表示 n 个进程已分配的各种资源数。

初始化函数：

void inputdata()

数据展示：

void showdata()

分配资源：

void changdata(int k)

恢复现场：

void rstordata(int k)

检查是否安全：

int chkerr(int s)

算法主体：

void bank()

## 二、算法流程

当一个进程提出资源申请时，银行家算法执行下列步骤以决定是否向其分配资源：

1）检查该进程所需要的资源是否已超过它所宣布的最大值。

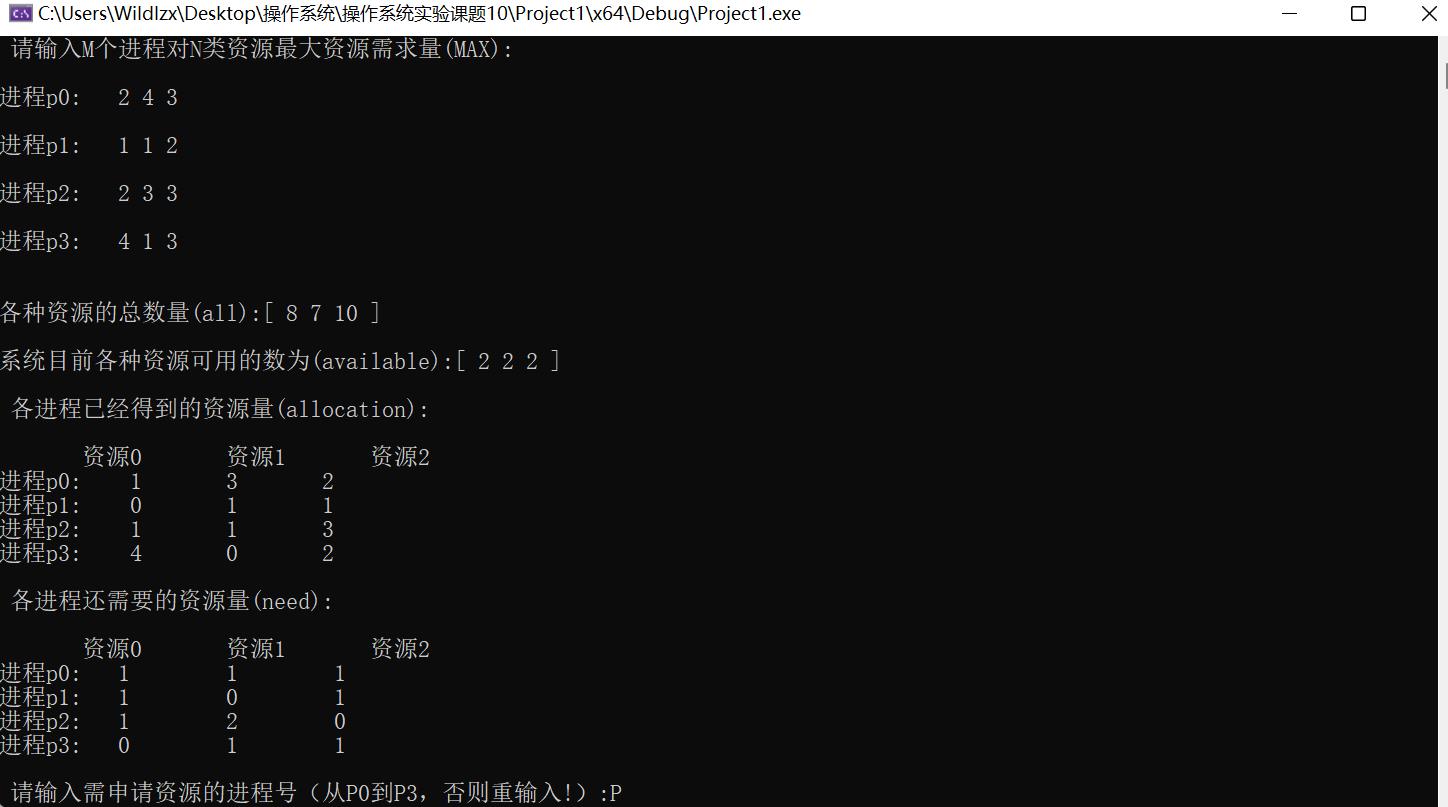
2）检查系统当前是否有足够资源满足该进程的请求。

3）系统试探着将资源分配给该进程，得到一个新状态。

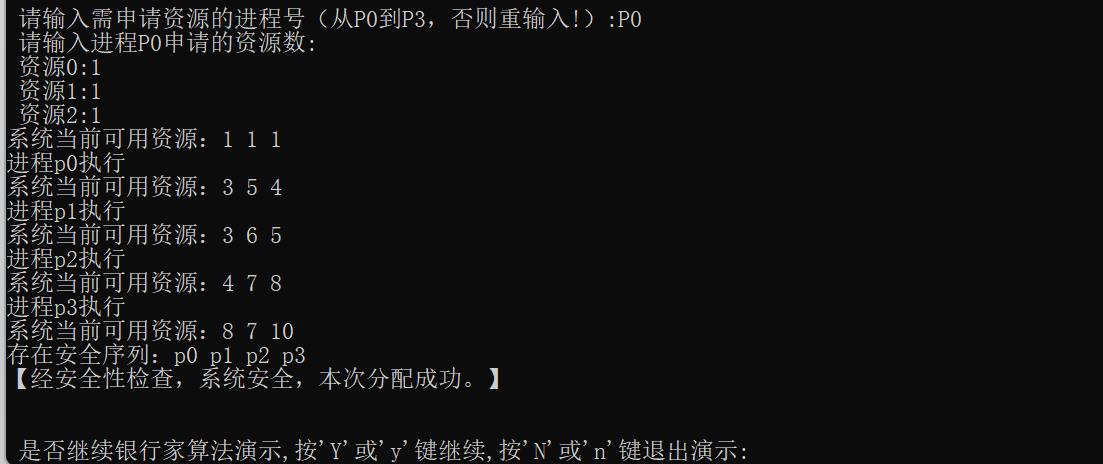
4）执行安全性算法，若该新状态是安全的，则分配完成；若新状态是不安全的，则恢复原状态，阻塞该进程。

## 三、测试结果及分析

首先对系统各类资源配备进行初始化：



然后进行资源申请操作，进行算法演示。



# 四、实验总结

本次实验主要进行了基于C语言对银行家算法的实现。在本次实验中，我更加深刻地理解了了银行家算法的流程。并且在具体编码实现时，遇到了很多问题和BUG，通过自己DUBUG和参考网络上经典的代码最终成功完成了实验，并在这个过程中学会了很多新的知识和技能。

# 五、源代码附录

1. #define \_CRT\_SECURE\_NO\_WARNINGS // VS用scanf()函数才需要的定义
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <string.h>
7. **int** M; //总进程数
8. **int** N; //资源种类
9. **int** ALL\_RESOURCE[10];//各种资源的数目总和
10. **int** MAX[10][10]; //M个进程对N类资源最大资源需求量
11. **int** AVAILABLE[10]; //系统可用资源数
12. **int** ALLOCATION[10][10]; //M个进程已经得到N类资源的资源量
13. **int** NEED[10][10]; //M个进程还需要N类资源的资源量
14. **int** Request[10]; //请求资源个数
15. **void** inputdata() {
16. **int** i, j;
17. printf("请输入总进程数:");
18. scanf("%d", &M);
19. printf("请输入资源种类数:");
20. scanf("%d", &N);
21. printf("请输入各种资源的总数量(all):");
22. **for** (j = 0; j < N; j++) {
23. scanf("%d", &ALL\_RESOURCE[j]);
24. }
26. printf(" 请输入各进程已经得到的资源量(allocation): \n\n");
27. **for** (i = 0; i < M; i++)
28. {
29. printf("进程p%d:    ", i);
30. **for** (j = 0; j < N; j++) {
31. scanf("%d", &ALLOCATION[i][j]);
32. }
33. printf("\n");
34. }
35. printf("\n");
36. printf(" 请输入M个进程对N类资源最大资源需求量(MAX):\n\n");
37. **for** (i = 0; i < M; i++) {
38. **for** (i = 0; i < M; i++)
39. {
40. printf("进程p%d:   ", i);
41. **for** (j = 0; j < N; j++) {
42. scanf("%d", &MAX[i][j]);
43. }
44. printf("\n");
45. }
46. }
48. printf("\n");
49. }
51. **void** showdata() //函数showdata,输出资源分配情况
52. {
53. **int** i, j;
54. printf("各种资源的总数量(all):[");
55. **for** (j = 0; j < N; j++) {
56. printf(" %d", ALL\_RESOURCE[j]);
57. }
58. printf(" ]\n\n");
60. printf("系统目前各种资源可用的数为(available):[");
62. **for** (j = 0; j < N; j++) {
63. printf(" %d", AVAILABLE[j]);
64. }
65. printf(" ]\n\n");
67. printf(" 各进程已经得到的资源量(allocation): \n\n");
68. **for** (j = 0; j < N; j++) {
69. printf("       资源%d", j);
70. }
71. printf("\n");
72. **for** (i = 0; i < M; i++)
73. {
74. printf("进程p%d:    ", i);
75. **for** (j = 0; j < N; j++) {
76. printf("%d       ", ALLOCATION[i][j]);
77. }
78. printf("\n");
79. }
80. printf("\n");
81. printf(" 各进程还需要的资源量(need):\n\n");
82. **for** (j = 0; j < N; j++) {
83. printf("       资源%d", j);
84. }
85. printf("\n");
86. **for** (i = 0; i < M; i++) {
87. **for** (i = 0; i < M; i++)
88. {
89. printf("进程p%d:   ", i);
90. **for** (j = 0; j < N; j++) {
91. printf("%d        ", NEED[i][j]);
92. }
93. printf("\n");
94. }
95. }
97. printf("\n");
98. }
100. **void** changdata(**int** k) //分配资源
101. {
102. **int** j;
103. **for** (j = 0; j < N; j++)
104. {
105. AVAILABLE[j] = AVAILABLE[j] - Request[j];
106. ALLOCATION[k][j] = ALLOCATION[k][j] + Request[j];
107. NEED[k][j] = NEED[k][j] - Request[j];
108. }
109. }
111. **void** rstordata(**int** k) //恢复现场
112. {
113. **int** j;
114. **for** (j = 0; j < N; j++)
115. {
116. AVAILABLE[j] = AVAILABLE[j] + Request[j];
117. ALLOCATION[k][j] = ALLOCATION[k][j] - Request[j];
118. NEED[k][j] = NEED[k][j] + Request[j];
119. }
120. }

123. **int** chkerr(**int** s) //函数chkerr,检查是否安全
124. {
125. **int** i = s;
126. **int** flag = 0;
127. **int**\* FINISH = (**int**\*)malloc(**sizeof**(**int**) \* M);
128. **int**\* WORK = (**int**\*)malloc(**sizeof**(**int**) \* N);
129. **int**\* array = (**int**\*)malloc(**sizeof**(**int**) \* M);
130. **int** cnt = 0;
131. **int** j;
133. memset(FINISH, 0, **sizeof**(FINISH));
134. memset(WORK, 0, **sizeof**(WORK));
135. printf("系统当前可用资源：");
136. **for** (j = 0; j < N; j++)
137. {
138. WORK[j] = AVAILABLE[j]; // 各分配的资源回收
139. printf("%d ", WORK[j]);
140. }
141. printf("\n");
142. {
143. **int** j;
144. **for** (j = 0; j < N; j++) { // 资源检查
145. **if** (NEED[i][j] > AVAILABLE[j]) { // 进程i所需要的资源j能否被满足
146. flag = 1; // 有不满足的，就退出
147. **break**;
148. }
149. }
150. }
152. **if** (flag == 0) {
153. **int** j;
154. FINISH[i] = 1; //i号进程，可以执行
155. printf("进程p%d执行\n", i);
156. printf("系统当前可用资源：");
157. **for** (j = 0; j < N; j++)
158. {
159. WORK[j] += ALLOCATION[i][j]; // 各分配的资源回收
160. printf("%d ", WORK[j]);
161. }
162. printf("\n");
163. array[cnt++] = i;
164. }
166. **while** (1) {
167. **int** i;
168. **int** flag1 = 0;
170. **for** (i = 0; i < M; ) {
171. **int** sum = 0;
172. **int** k;
173. **for** (k = 0; k < M; k++) {
174. sum += FINISH[k];
175. }
176. **if** (sum == M) {
177. **int** ij;
178. printf("存在安全序列：");
179. **for** (ij = 0; ij < M; ij++) {
180. printf("p%d ", array[ij]);
181. }
182. printf("\n");
183. printf("【经安全性检查，系统安全，本次分配成功。】\n");
184. printf("\n");
185. **return** 1;
186. }
188. **if** (FINISH[i] != 1) {
189. flag1 = 0;
190. **for** (**int** j = 0; j < N; j++) { // 资源检查
191. **if** (NEED[i][j] > WORK[j]) { // 进程i所需要的资源j能否被满足
192. flag1 = 1; // 有不满足的，就退出
193. **break**;
194. }
195. }
196. }
198. **if** (flag1 == 0 && FINISH[i] != 1) { // 表示i进程没执行，且可以执行
199. **int** j;
200. FINISH[i] = 1; //i号进程，可以执行
201. printf("进程p%d执行\n", i);
202. printf("系统当前可用资源：");
203. **for** (j = 0; j < N; j++)
204. {
205. WORK[j] += ALLOCATION[i][j]; // 各分配的资源回收
206. printf("%d ", WORK[j]);
207. }
208. printf("\n");
209. array[cnt++] = i;
210. i = 0;
211. }
212. **else** { // 不能执行判断下一个进程
213. i++;
214. }
215. }
216. printf("\n");
217. printf("【系统不安全!!! 本次资源申请不成功!!!】\n");
218. printf("\n");
219. **return** 0;
221. }
222. }
224. **void** bank()   //银行家算法主体
225. {
226. **int** i = 0, j = 0;
227. **char** flag = 'Y';
229. **while** (flag == 'Y' || flag == 'y')
230. {
231. i = -1;
232. **while** (i < 0 || i >= M)
233. {
234. printf(" 请输入需申请资源的进程号（从P0到P%d，否则重输入!）:", M - 1);
235. printf("P");
236. scanf("%d", &i);
237. **while** (i < 0 || i >= M) {
238. printf(" 输入的进程号不存在，重新输入!\n");
239. printf(" 请输入需申请资源的进程号（从P0到P%d，否则重输入!）:", M - 1);
240. printf("P");
241. scanf("%d", &i);
242. }
243. printf(" 请输入进程P%d申请的资源数:\n", i);
244. **for** (j = 0; j < N; j++)
245. {
246. printf(" 资源%d:", j);
247. scanf("%d", &Request[j]);
248. **if** (Request[j] > NEED[i][j]) //若请求的资源数大于进程还需要i类资源的资源量j
249. {
250. printf(" 进程P%d申请的资源数大于进程P%d还需要%d类资源的资源量!", i, i, j);
251. printf("申请不合理，出错!请重新选择!\n\n");
252. flag = 'N';
253. **break**;
254. }
255. **else**
256. {
257. **if** (Request[j] > AVAILABLE[j]) //若请求的资源数大于可用资源数
258. {
259. printf(" 进程P%d申请的资源数大于系统可用%d类资源的资源量!", i, j);
260. printf("申请不合理，出错!请重新选择!\n\n");
261. flag = 'N';
262. **break**;
263. }
264. }
265. }
266. **if** (flag == 'Y' || flag == 'y')
267. {
268. changdata(i); //调用changdata(i)函数，改变资源数
269. **if** (chkerr(i)) //若系统安全
270. {
272. printf("\n");
273. printf(" 是否继续银行家算法演示,按'Y'或'y'键继续,按'N'或'n'键退出演示: ");
274. scanf("%\*c%c", &flag);
275. rstordata(i); //调用rstordata(i)函数，恢复资源数
276. showdata();   //输出资源分配情况
277. }
278. **else** {      //若系统不安全
280. printf("\n");
281. printf(" 是否继续银行家算法演示,按'Y'或'y'键继续,按'N'或'n'键退出演示: ");
283. scanf("%\*c%c", &flag);
284. showdata(); //输出资源分配情况
285. }
286. }
287. **else**      //若flag=N||flag=n
288. {
289. printf("\n");
290. printf(" 是否继续银行家算法演示,按'Y'或'y'键继续,按'N'或'n'键退出演示: ");
292. scanf("%\*c%c", &flag);
293. showdata();
294. }
296. }
297. }
298. }
300. //主函数
301. **int** main()
302. {
303. **int** i = 0, j = 0, p;
304. //初始化资源数量
305. inputdata();
306. **for** (j = 0; j < N; j++)
307. {
308. p = ALL\_RESOURCE[j];
309. **for** (i = 0; i < M; i++)
310. {
311. p = p - ALLOCATION[i][j];//减去已经被占据的资源
312. AVAILABLE[j] = p;
313. **if** (AVAILABLE[j] < 0)
314. AVAILABLE[j] = 0;
315. }
316. }
317. **for** (i = 0; i < M; i++)
318. {
319. **for** (j = 0; j < N; j++) {
320. NEED[i][j] = MAX[i][j] - ALLOCATION[i][j];
321. }
322. }
324. showdata();
325. bank();
327. **return** 0;
328. }