#### Operating system experiment report

1. **Name:** Luoyu Mei **Number:** 71117408 **Date:** 21/05/2019
2. **Working target:** Handle banker algorithm using，create multi thread to make satisfy critical section problem.

**Working environment:** as basic operating system using G++ version 8.2.0 on Windows to handle banker algorithm.

1. **Steps:**
2. Define available, allocation, max array, create basic banker algorithm.
3. In order to make the program satisfy , I create five thread function respectively represent the application of system resource
4. Write safety function using banker algorithm to check weather the application is safe, if safe then output safety queue, else reject the request.
5. Get input of the need of process, using safety function to get the safety queue.

**4. Flow chart:**

begin

Output the situation of banker algorithm now.

Get input of thread’s need array and create initial thread.

Using banker algorithm to check weather the need can be satisfy

end

If the need can be satisfied, output the safety queue

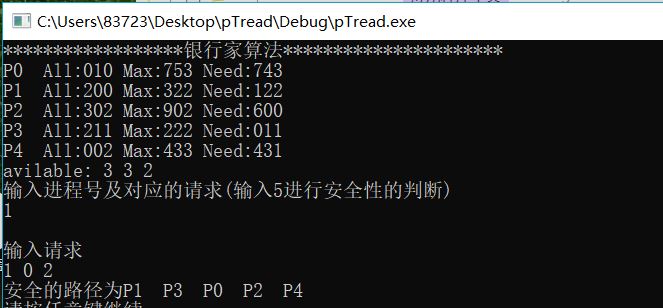
**5. Main data structure:**

I use five thread functions for five process, the available resource, allocation, max source of different process in system is saved using three arrays. After getting need queue form user, the program will firstly make sure it is safety using banker algorithm, when it is satisfy then output the safe queue.

**The code file of my program will also be bale together with this report.**

**6. Experiment result:**

Windows：



**7. Filling of experiment:**

It isn’t so difficult for me to use Windows API to imply multi thread and make sure the critical section is accessed respectively. However, in order to use banker algorithm I need to write five function for five process and only change the allocation, available queue in system when the apply is safe.

**8. Code for windows:**

1. #include <iostream>
2. #include <process.h>
3. #include <Windows.h>
5. **using** **namespace** std;
6. //定义5个线程函数，分别表示5个进程对系统资源的请求
8. **DWORD** WINAPI pro0(**PVOID** pvPram);
10. **DWORD** WINAPI pro1(**PVOID** pvPram);
12. **DWORD** WINAPI pro2(**PVOID** pvPram);
14. **DWORD** WINAPI pro3(**PVOID** pvPram);
16. **DWORD** WINAPI pro4(**PVOID** pvPram);
17. //申明安全函数
19. **void** ifsafe();
21. **int** available[3] = { 3, 3, 2 }; //系统当前可用的资源数
22. //5个进程已分配的资源数
24. **int** alloc[5][3] = { {0, 1, 0},
25. {2, 0, 0},
26. {3, 0, 2},
27. {2, 1, 1},
28. {0, 0, 2}
29. };
30. //5个进程的最大资源数
32. **int** max[5][3] = { {7, 5, 3},
33. {3, 2, 2},
34. {9, 0, 2},
35. {2, 2, 2},
36. {4, 3, 3}
37. };
39. **int** a, b, c, d; //用于用户的输入b,c,d表示进程的请求资源数
41. **int** main()
42. {
43. //以一定的格式输出当前的资源分配情况
44. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*银行家算法\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
45. cout << "P0 " << " All:" << alloc[0][0] << alloc[0][1] << alloc[0][2] << " Max:" << max[0][0] << max[0][1] << max[0][2] << " Need:" << max[0][0] - alloc[0][0] << max[0][1] - alloc[0][1] << max[0][2] - alloc[0][2] << endl;
46. cout << "P1 " << " All:" << alloc[1][0] << alloc[1][1] << alloc[1][2] << " Max:" << max[1][0] << max[1][1] << max[1][2] << " Need:" << max[1][0] - alloc[1][0] << max[1][1] - alloc[1][1] << max[1][2] - alloc[1][2] << endl;
47. cout << "P2 " << " All:" << alloc[2][0] << alloc[2][1] << alloc[2][2] << " Max:" << max[2][0] << max[2][1] << max[2][2] << " Need:" << max[2][0] - alloc[2][0] << max[2][1] - alloc[2][1] << max[2][2] - alloc[2][2] << endl;
48. cout << "P3 " << " All:" << alloc[3][0] << alloc[3][1] << alloc[3][2] << " Max:" << max[3][0] << max[3][1] << max[3][2] << " Need:" << max[3][0] - alloc[3][0] << max[3][1] - alloc[3][1] << max[3][2] - alloc[3][2] << endl;
49. cout << "P4 " << " All:" << alloc[4][0] << alloc[4][1] << alloc[4][2] << " Max:" << max[4][0] << max[4][1] << max[4][2] << " Need:" << max[4][0] - alloc[4][0] << max[4][1] - alloc[4][1] << max[4][2] - alloc[4][2] << endl;
50. cout << "avilable: " << available[0] << " " << available[1] << " " << available[2] << endl;
51. cout << "输入进程号及对应的请求(输入5进行安全性的判断)" << endl;
52. cin >> a;
53. cout << endl;
54. **if** (a == 5)
55. {
56. ifsafe();
57. }
58. **else**
59. {
60. cout << "输入请求" << endl;
61. cin >> b >> c >> d;
62. //根据进程号来启用相应得进程
63. **switch** (a)
64. {
65. **case** 0:
66. {
67. **HANDLE** pr0 = CreateThread(NULL, 0,
68. pro0, NULL, 0,
69. NULL);
70. CloseHandle(pr0);
71. **break**;
72. }
73. **case** 1:
74. {
75. **HANDLE** pr1 = CreateThread(NULL, 0,
76. pro1, NULL, 0,
77. NULL);
78. CloseHandle(pr1);
79. **break**;
80. }
81. **case** 2:
82. {
83. **HANDLE** pr2 = CreateThread(NULL, 0,
84. pro2, NULL, 0,
85. NULL);
86. CloseHandle(pr2);
87. **break**;
88. }
89. **case** 3:
90. {
91. **HANDLE** pr3 = CreateThread(NULL, 0,
92. pro3, NULL, 0,
93. NULL);
94. CloseHandle(pr3);
95. **break**;
96. }
97. **case** 4:
98. {
99. **HANDLE** pr4 = CreateThread(NULL, 0,
100. pro4, NULL, 0,
101. NULL);
102. CloseHandle(pr4);
103. **break**;
104. }
105. }
106. }
107. system("pause");
108. **return** 0;
109. }
111. **DWORD** WINAPI pro0(**PVOID** pvPram)
112. {
113. //已分配的资源加上请求的资源数
114. alloc[0][0] += b;
115. alloc[0][1] += c;
116. alloc[0][2] += d;
117. //系统可用的资源数减去请求的资源数
118. available[0] -= b;
119. available[1] -= c;
120. available[2] -= d;
121. **int** x = 0;
122. **for** (**int** i = 0; i < 3; i++)
123. {
124. **if** (available[i] < 0) //判断请求资源数是否大于系统可用的资源数
125. {
126. cout << "请求资源大于可用资源" << endl;
127. }
128. **else**
129. {
130. x++;
131. }
132. }
133. **if** (x == 3)
134. //如果三种类型的请求资源数都小于系统可用的资源数，则进行安全算法的判断
135. ifsafe();
136. **return** 0;
137. }
139. **DWORD** WINAPI pro1(**PVOID** pvPram)
140. {
141. alloc[1][0] += b;
142. alloc[1][1] += c;
143. alloc[1][2] += d;
144. available[0] -= b;
145. available[1] -= c;
146. available[2] -= d;
147. **int** x = 0;
148. **for** (**int** i = 0; i < 3; i++)
149. {
150. **if** (available[i] < 0)
151. {
152. cout << "请求资源大于可用资源" << endl;
153. }
154. **else**
155. {
156. x++;
157. }
158. }
159. **if** (x == 3)
160. ifsafe();
161. **return** 0;
162. }
164. **DWORD** WINAPI pro2(**PVOID** pvPram)
165. {
166. alloc[2][0] += b;
167. alloc[2][1] += c;
168. alloc[2][2] += d;
169. available[0] -= b;
170. available[1] -= c;
171. available[2] -= d;
172. **int** x = 0;
173. **for** (**int** i = 0; i < 3; i++)
174. {
175. **if** (available[i] < 0)
176. {
177. cout << "请求资源大于可用资源" << endl;
178. }
179. **else**
180. {
181. x++;
182. }
183. }
184. **if** (x == 3)
185. ifsafe();
186. **return** 0;
187. }
189. **DWORD** WINAPI pro3(**PVOID** pvPram)
190. {
191. alloc[3][0] += b;
192. alloc[3][1] += c;
193. alloc[3][2] += d;
194. available[0] -= b;
195. available[1] -= c;
196. available[2] -= d;
197. **int** x = 0;
198. **for** (**int** i = 0; i < 3; i++)
199. {
200. **if** (available[i] < 0)
201. {
202. cout << "请求资源大于可用资源" << endl;
203. }
204. **else**
205. {
206. x++;
207. }
208. }
209. **if** (x == 3)
210. ifsafe();
211. **return** 0;
212. }
214. **DWORD** WINAPI pro4(**PVOID** pvPram)
215. {
216. alloc[4][0] += b;
217. alloc[4][1] += c;
218. alloc[4][2] += d;
219. available[0] -= b;
220. available[1] -= c;
221. available[2] -= d;
222. **int** x = 0;
223. **for** (**int** i = 0; i < 3; i++)
224. {
225. **if** (available[i] < 0)
226. {
227. cout << "请求资源大于可用资源" << endl;
228. }
229. **else**
230. {
231. x++;
232. }
233. }
234. **if** (x == 3)
235. ifsafe();
236. **return** 0;
237. }
239. **void** ifsafe()
240. {
241. **bool** finish[5] = { **false** }; //每个进程当前的状态设为false表示该进程没有释放资源
242. **int** n[5] = { 0 }; //记录数组，如果该进程need数小于available，则将该进程号储存在数组中，最后用于输出
243. **int** j = 0; //进行进程的遍历，初值为0
244. **int** q = 0; //计数器用于计数进程状态为true的个数
245. **int** nn = 0; //
246. **for** (; j < 5; j++)
247. {
248. **int** m = 0; //计数器，用于下面计数三种资源有几个满足要求
249. **if** (finish[j] == **true**)
250. {
251. **continue**;
252. //每次循环前要判断该进程是否已经完成了资源的释放，如果已经释放，则跳过
253. }
254. **else**
255. {
256. **for** (**int** i = 0; i < 3; i++)
257. {
258. **if** ((max[j][i] - alloc[j][i]) <= available[i]) //需求的资源小于系统可用的资源数
259. {
260. m++;
261. }
262. }
263. **if** (m == 3) //当m=3说明系统可以满足该进程的需求
264. {
265. n[nn] = j; //记录进程号
266. finish[j] = **true**; //释放该进程
267. **for** (**int** k = 0; k < 3; k++)
268. {
269. available[k] += alloc[j][k]; //系统可用资源的回收
270. }
271. j = -1; //将j=-1是为了下一次进行进程的遍历时，要从头开始，防止前面的一些进程当时不满足要求但现在可以了
272. nn++;
273. //记录数组下一次记录要从下一个位子开始
274. }
275. }
276. }
277. **for** (**int** i = 0; i < 5; i++)
278. {
279. **if** (finish[i] == **false**)
280. {
281. cout << "不存在安全算法" << endl;
282. //只要5个进程有一个没有通过而释放资源，说明当前系统不安全
283. }
284. **else**
285. {
286. q++;
287. }
288. }
289. **if** (q == 5) //如果q=5说明系统所以进程都释放了资源，系统安全
290. {
291. cout << "安全的路径为";
292. **for** (**int** i = 0; i < 5; i++)
293. {
294. cout << "P" << n[i] << "  ";      //一个一个释放记录数组中记录的进程号
295. }
296. cout << endl;
297. }
298. }