浙江工业大学

操作系统原理实验报告

(2021 级)

实验二：银行家算法实验

学生姓名：温家伟

学生学号：202103151422

学科专业：大数据分析 2101 班

所在学院：理学院

提交日期：2023 年 12 月 3 日

目录

1 实验目的 **2**

2 实验要求 **2**

3 源代码 **2**

4 实验结果 **10**

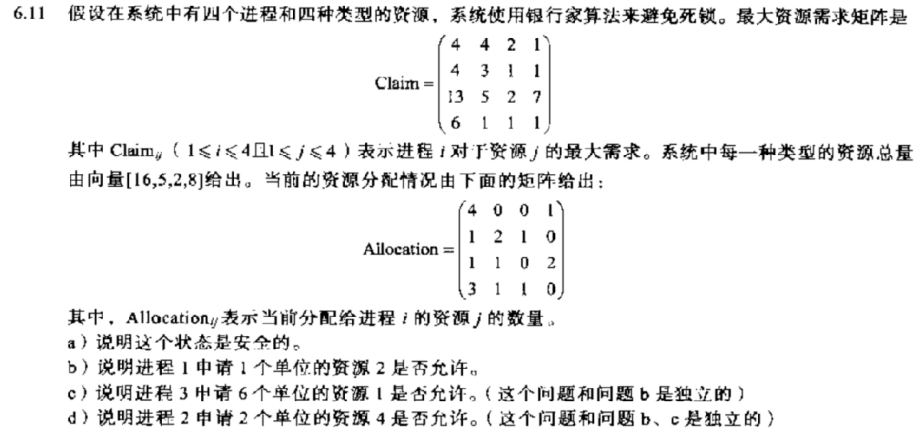
1 实验目的

根据银行家算法的思想，编写程序，解决并发进程的死锁问题。

2 实验要求

本实验要求设计并实现银行家算法。银行家算法是死锁避免的经典算法，其核心思想是：进程动态地申请资源，每次申请资源时系统都执行安全状态检查算法判断本次申请是否会造成系统处于不安全状态，如果不安全则阻塞进程；如果安全状态，则完成资源分配。安全状态检查

算法的思想是找到一个安全序列，使所有进程都能执行完毕。如果找到，则处于安全状态，否则为不安全状态。



6.11 假设在系统中有四个进程和四种类型的资源，系统使用银行家算法来避免死锁。最大资源需求矩阵是

$$\text{Claim} = \begin{pmatrix} 4 & 4 & 2 & 1 \\ 4 & 3 & 1 & 1 \\ 13 & 5 & 2 & 7 \\ 6 & 1 & 1 & 1 \end{pmatrix}$$

其中 Claim_{ij}（$1 \leqslant i \leqslant 4$ 且 $1 \leqslant j \leqslant 4$）表示进程 i 对于资源 j 的最大需求。系统中每一种类型的资源总量由向量[16,5,2,8]给出。当前的资源分配情况由下面的矩阵给出：

$$\text{Allocation} = \begin{pmatrix} 4 & 0 & 0 & 1 \\ 1 & 2 & 1 & 0 \\ 1 & 1 & 0 & 2 \\ 3 & 1 & 1 & 0 \end{pmatrix}$$

其中，Allocation_{ij} 表示当前分配给进程 i 的资源 j 的数量。

a）说明这个状态是安全的。

b）说明进程 1 申请 1 个单位的资源 2 是否允许。

c）说明进程 3 申请 6 个单位的资源 1 是否允许。（这个问题和问题 b 是独立的）

d）说明进程 2 申请 2 个单位的资源 4 是否允许。（这个问题和问题 b、c 是独立的）

图 1: 实验要求

3 源代码

```
1
2 #include <iostream>
3 using namespace std;
4
5 const int p = 4; // 进程数
```

```
6  const int r = 4; // 资源种类
7
8  // 比较函数
9  int com(int m[r], int n[r]) {
10     int i, flag = 0;
11     for (i = 0; i < r; i++) {
12         if (m[i] < n[i]) {
13             flag = 1;
14             break;
15         }
16     }
17     if (flag == 1) {
18         return 0;
19     } else {
20         return 1;
21     }
22 }
23
24 int test(int need[r]) {
25     int i, a, flag = 0;
26     for (i = 0, a = 0; i < r; i++) {
27         if (need[i] == 0) {
28             a++;
29         }
30     }
31     if (a == r) {
32         return 1;
33     } else {
34         return 0;
35     }
36 }
37
38 // 安全性检验函数
```

```
39 int stest(int allocations[p][r], int needs[p][r], int
       availables[r]) {
40     int i, j, k, l, flag = 0;
41     int need[r], finish[p], available[r];
42     for (i = 0; i < p; i++) {
43         finish[i] = 0; //
               finish为1即表示available满足某一进程并让其实现
44     }
45
46     cout << "初始available" << '\t';
47     for (i = 0; i < r; i++) {
48         available[i] = availables[i];
49         cout << availables[i] << '\t';
50     } // 保存数据
51     cout << "\n分配序列: \n";
52     for (k = 1; k <= p; k++) { //
           全搜索, 直至实现或不可能实现
53         for (i = 0; i < p; i++) {
54             if (finish[i] == 1) {
55                 continue;
56             } else {
57                 for (j = 0; j < r; j++) {
58                     need[j] = needs[i][j];
59                 }
60                 if (com(available, need)) {
61
62                     cout << "\n第" << k <<
                           "次分配进程为:";
63                     finish[i] = 1;
64                     cout << i + 1 << '\t';
65                     cout << "  此时的available: " <<
                           '\t';
66                     for (l = 0; l < r; l++) {
```

```
67                          available[l] = available[l] +
                               allocations[i][l];
68                          cout << available[l] << '\t';
69                      }
70                      break;
71                  }
72              }
73          }
74      }
75      cout << endl;
76      for (l = 0; l < p; l++) {
77          if (finish[l] == 0) {
78              flag = 1;
79          }
80      }
81      if (flag == 0) {
82          return 1; //
                flag为记录finish是否有0存在的标记，当flag=0时，安全
83      } else {
84          return 0;
85      }
86  }
87
88  // 申请进程后的安全性检验函数
89  void rtest(int b[p][r], int c[p][r], int d[r], int
       req[r], int n) { //
       req-request, n-第n个进程申请资源
90      int i, j;
91      int t[r];
92      n = n - 1;
93      for (i = 0; i < r; i++) {
94          t[i] = c[n][i];
95      }
```

```
96      if (com(t, req)) { // 对request和need进行比较
97          cout << "第" << n + 1 <<
                "个进程申请资源通过(request<=need)\n";
98
99          if (com(d, req)) { //
                对request和available进行比较
100             for (j = 0; j < r; j++) {
101                 b[n][j] = b[n][j] + req[j];
102                 c[n][j] = c[n][j] - req[j];
103                 d[j] = d[j] - req[j];
104             }
105             if (stest(b, c, d)) {
106                 cout << "第" << n + 1 <<
                        "个进程申请资源成功(request<=available)\n";
107                 int a = test(d);
108                 if (a == 1) {
109                     for (i = 0; i < r; i++) {
110                         d[i] = b[n][i] + d[i]; //
                                资源回归available
111                     }
112                 }
113             } else {
114                 cout << "第" << n + 1 <<
                        "个进程申请资源失败，恢复以前状态。安全队列寻找失败\n";
115                 for (j = 0; j < r; j++) {
116                     b[n][j] = b[n][j] - req[j];
117                     c[n][j] = c[n][j] + req[j];
118                     d[j] = d[j] + req[j];
119                 }
120             }
121         } else {
122             cout <<
                    "进程等待…(request>available)…\n";
```

```
123         }
124     } else {
125         cout << "第" << n + 1 <<
                "个进程申请资源失败，恢复以前状态。安全队列寻找失败\n";
126     }
127 }
128
129
130 int main()
131 {
132     int j, n, i, t; // n-第n个资源申请
133     int flag1 = 1;  // flag1作为循环条件
134     int ALL[4] = {16, 5, 2, 8};
135     int max[4][4] = {4, 4, 2, 1, 4, 3, 1, 1, 13, 5,
            2, 7, 6, 1, 1, 1};
136     int allocation[4][4] = {4, 0, 0, 1, 1, 2, 1, 0,
            1, 1, 0, 2, 3, 1, 1, 0};
137     int need[4][4] = {0, 4, 2, 0, 3, 1, 0, 1, 12, 4,
            2, 5, 3, 0, 0, 1};
138     int available[4];
139     int request[4];
140
141     for (i = 0; i < r; i++)
142     {
143         t = 0;
144         for (j = 0; j < p; j++)
145         {
146             t = t + allocation[j][i];
147         }
148         available[i] = ALL[i] - t;
149     }
150     //  input(max, allocation, c, available);
151     printf("max\tallocation\tneed\tavailable\n");
```

```
152     for (i = 0; i < p; i++)
153     {
154         for (j = 0; j < r; j++)
155         {
156             cout << max[i][j] << " ";
157         }
158         cout << '\t';
159         for (j = 0; j < r; j++)
160         {
161             cout << allocation[i][j] << " ";
162         }
163         cout << '\t' << '\t';
164         for (j = 0; j < r; j++)
165         {
166             cout << need[i][j] << " ";
167         }
168         cout << '\t';
169         if (i == 0)
170         {
171             for (j = 0; j < r; j++)
172             {
173                 cout << available[j] << " ";
174             }
175         }
176         cout << endl;
177     }
178
179     cout << endl;
180     if (stest(allocation, need, available) == 1)
181         cout << "初始状态安全！\n";
182     else
183     {
184         cout << "初始状态不安全，请仔细检查！\n";
```

```
185         return 0;
186     }
187
188     while (flag1)
189     {
190         cout << "输入申请资源的进程的序号: \n";
191         cin >> n;
192
193         cout << "输入request数据: \n";
194         for (j = 0; j < r; j++)
195             cin >> request[j];
196
197         rtest(allocation, need, available, request,
                n);
198         printf("max\tallocation\tneed\tavailable\n");
199         for (i = 0; i < p; i++)
200         {
201             for (j = 0; j < r; j++)
202             {
203                 cout << max[i][j] << " ";
204             }
205             cout << '\t';
206             for (j = 0; j < r; j++)
207             {
208                 cout << allocation[i][j] << " ";
209             }
210             cout << '\t' << '\t';
211             for (j = 0; j < r; j++)
212             {
213                 cout << need[i][j] << " ";
214             }
215             cout << '\t';
216             if (i == 0)
```

```
217             {
218                 for (j = 0; j < r; j++)
219                 {
220                     cout << available[j] << " ";
221                 }
222             }
223             cout << endl;
224         }
225         cout << "\n继续分配输入1，退出输入0：\n";
226         cin >> flag1;
227     }
228 
229     return 0;
230 }
```

4 实验结果

```
[bbjsxl@VM-20-8-centos code7]$ make
g++ -std=c++11 test.cc -o test -lpthread
[bbjsxl@VM-20-8-centos code7]$ ./test
max     allocation      need    available
4 4 2 1         4 0 0 1                 0 4 2 0         7 1 0 5
4 3 1 1         1 2 1 0                 3 1 0 1
13 5 2 7        1 1 0 2                 12 4 2 5
6 1 1 1         3 1 1 0                 3 0 0 1

初始available   7       1       0       5
分配序列:

第1次分配进程为:2          此时的available:      8       3       1       5
第2次分配进程为:4          此时的available:      11      4       2       5
第3次分配进程为:1          此时的available:      15      4       2       6
第4次分配进程为:3          此时的available:      16      5       2       8
初始状态安全!
输入申请资源的进程的序号:
1
输入request数据:
0 1 0 0
第1个进程申请资源通过(request<=need)
初始available   7       0       0       5
分配序列:

第1次分配进程为:4          此时的available:      10      1       1       5
第2次分配进程为:2          此时的available:      11      3       2       5
第3次分配进程为:1          此时的available:      15      4       2       6
第4次分配进程为:3          此时的available:      16      5       2       8
第1个进程申请资源成功(request<=available)
max     allocation      need    available
4 4 2 1         4 1 0 1                 0 3 2 0         7 0 0 5
4 3 1 1         1 2 1 0                 3 1 0 1
13 5 2 7        1 1 0 2                 12 4 2 5
6 1 1 1         3 1 1 0                 3 0 0 1
```

图 2： 实验运行结果图

```
[bbjsxl@VM-20-8-centos code7]$ ./test
max     allocation      need    available
4 4 2 1         4 0 0 1                 0 4 2 0         7 1 0 5
4 3 1 1         1 2 1 0                 3 1 0 1
13 5 2 7        1 1 0 2                 12 4 2 5
6 1 1 1         3 1 1 0                 3 0 0 1

初始available   7       1       0       5
分配序列:

第1次分配进程为:2          此时的available:      8       3       1       5
第2次分配进程为:4          此时的available:      11      4       2       5
第3次分配进程为:1          此时的available:      15      4       2       6
第4次分配进程为:3          此时的available:      16      5       2       8
初始状态安全!
输入申请资源的进程的序号:
3
输入request数据:
6 0 0 0
第3个进程申请资源通过(request<=need)
初始available   1       1       0       5
分配序列:

第3个进程申请资源失败，恢复以前状态。安全队列寻找失败
max     allocation      need    available
4 4 2 1         4 0 0 1                 0 4 2 0         7 1 0 5
4 3 1 1         1 2 1 0                 3 1 0 1
13 5 2 7        1 1 0 2                 12 4 2 5
6 1 1 1         3 1 1 0                 3 0 0 1
```

图 3： 实验运行结果图

```
[bbjsxl@VM-20-8-centos code7]$ ./test
max     allocation      need    available
4 4 2 1         4 0 0 1                 0 4 2 0         7 1 0 5
4 3 1 1         1 2 1 0                 3 1 0 1
13 5 2 7        1 1 0 2                 12 4 2 5
6 1 1 1         3 1 1 0                 3 0 0 1

初始available   7       1       0       5
分配序列：

第1次分配进程为:2          此时的available:      8       3       1       5
第2次分配进程为:4          此时的available:      11      4       2       5
第3次分配进程为:1          此时的available:      15      4       2       6
第4次分配进程为:3          此时的available:      16      5       2       8
初始状态安全！
输入申请资源的进程的序号：
2
输入request数据：
0 0 0 2
第2个进程申请资源失败，恢复以前状态。安全队列寻找失败
max     allocation      need    available
4 4 2 1         4 0 0 1                 0 4 2 0         7 1 0 5
4 3 1 1         1 2 1 0                 3 1 0 1
13 5 2 7        1 1 0 2                 12 4 2 5
6 1 1 1         3 1 1 0                 3 0 0 1
```

图 **4：** 实验运行结果图