

0.1 Project 06 银行家算法

0.1 Project 06 银行家算法

0.1.1 一、项目介绍

0.1.2 二、算法介绍

0.1.3 三、实现原理

0.1.3.1 1、数据读取

0.1.3.2 2、口令识别

0.1.3.3 3、核心算法一 资源请求

0.1.3.4 4、核心算法二 银行家算法

0.1.3.5 5、核心算法三 资源释放

0.1.3.6 6、源代码展示

0.1.4 四、效果演示

0.1.1 一、项目介绍

对于这个项目，你将编写一个程序来实现第 8.6.3 节中讨论的银行家算法。客户向银行请求和释放资源。只有当系统处于安全状态时，银行家才会批准请求。使系统处于不安全状态的请求将被拒绝。尽管描述此项目的代码示例以 C 语言说明，但也可以使用 Java 开发解决方案。

资源分配图算法不适用于每种资源类型有多个实例的资源分配系统。我们接下来描述的死锁避免算法适用于这样的系统，但效率低于资源分配图方案。这种算法通常被称为银行家算法。之所以选择这个名称，是因为该算法可以在银行系统中使用，以确保银行永远不会以无法再满足所有客户需求的方式分配其可用现金。当一个新线程进入系统时，它必须声明它可能需要的每种资源类型的的最大实例数。此数量不得超过系统中资源的总数。当用户请求一组资源时，系统必须确定这些资源的分配是否会使系统处于安全状态。如果可以，则分配资源；否则，线程必须等到其他线程释放足够的资源。必须维护几个数据结构来实现银行家的算法。这些数据结构对资源分配系统的状态进行编码。

0.1.2 二、算法介绍

根据我们所学的内容，银行家算法中有几个重要的参数：

- Allocation 对于进程目前已经分配的资源表
- Max，对于进程，执行完成所需要最大的资源数目
- Available 可以获得的资源的数目
- Need：用Max减去Allocation的对应条目，得到每个进程执行完还需要的资源的数目

在回顾了我们的之前所学的概念后，我们来开始用银行家算法解决问题：

- 首先，我们要计算出各个进程所需要的资源数目
- 然后我们要计算出当前Available的资源数目
- 我们再拿着可获得的资源数目，和每个进程Need的资源比较一下
- 一旦可以执行，我们就修改Available的资源数目。也就是说Available的资源要增加，增加上之前刚刚执行完成的进程的Allocation表，即可
- 这样一直进行下去，就可以得到一个序列，按照这个序列可以按顺序执行完成任务，不会死锁。

0.1.3 三、实现原理

0.1.3.1 1、数据读取

- 首先我们要能够根据要求，完成基本的数据读取工作。
- 根据教材的要求，我们在主函数的一开始就完成了数据的写入，例如 maximum 的数组，以及 available 的数组
- 考虑到文件的格式，所以中间的分隔符是 `,`，所以要调用 strsep 的库函数帮助完成分隔，同时转化为整数

```
1 // initialize array MAXIMUM with txt file
2 FILE *in;
3 in = fopen("max_need.txt", "r");
4 char customer[SIZE];
5 char *temp;
```

```

6      for(int i=0;i<NUMBER_OF_CUSTOMERS;++i)
7      {
8          fgets(customer,SIZE,in);
9          temp = strdup(customer);
10         for(int j=0;j<NUMBER_OF_RESOURCES;j++)
11             maximum[i][j]=need[i][j]=atoi(strsep(&temp, ",")), allocation[i]
12             [j]=0;
13     }
14
15     // initialize array **available** with txt file
16     in = fopen("available_resource.txt", "r");
17     char ava[SIZE];
18     fgets(ava, SIZE, in);
19     temp = strdup(ava);
20     for(int i=0;i<NUMBER_OF_RESOURCES;i++)
21         available[i]=atoi(strsep(&temp, ","));
22     fclose(in);

```

0.1.3.2 2、口令识别

- 为了整个程序能够正常的运行下去，也能正常的退出，根据教材制定了下面的运行逻辑。
 - 如果输入end，就表示退出
 - 如果输入RQ，就表示请求资源，当然后面要跟上一定的参数
 - 如果输入RL，就表示释放资源，当然也要跟上一定的参数
 - 如果输入的是非法的命令，那么也会提示报错信息。
- 简要的运行逻辑可以参见下面的代码逻辑，

```

1     while(ifrun)
2     {
3         printf("Banker >>> ");
4         fgets(input_line, SIZE, stdin);
5
6         if(strcmp(input_line, "end\n") == 0)
7         {
8             ifrun = 0;
9             printf("[Info] Banker's Algorithm Finished, exit!\n");
10            continue;
11        }
12
13        if(input_line[0] == 'R' && input_line[1] == 'Q'){
14            //decode instruction
15            temp = strdup(input_line);
16            strsep(&temp, " ");
17            for(int i=0; i<=NUMBER_OF_RESOURCES; ++i)

```

```

18         input_instruction[i] = atoi(strsep(&temp, " "));
19         request_value = request(input_instruction);
20         if(request_value!=0)
21             printf("Request Failed. Please Follow Rules.\n");
22         continue;
23     }
24
25     else if(input_line[0] == 'R' && input_line[1] == 'L'){
26         temp = strdup(input_line);
27         strsep(&temp," ");
28         for(int i=0; i<=NUMBER_OF_RESOURCES; ++i)
29             input_instruction[i] = atoi(strsep(&temp, " "));
30         release_value = release(input_instruction);
31         if(release_value!=0)
32             printf("Release Failed. Please Follow Rules.\n");
33         continue;
34     }
35
36     else if(input_line[0]=='*')
37         state_display();
38     else{
39         printf("[Error] Invalid Token, Please Input Again!\n");
40     }
41
42 }
```

0.1.3.3 3、核心算法一 资源请求

- 在处理用户的请求的时候，首先要校验用户的输入是否是合理的请求，下面的请求会被直接拒绝不被执行
 - 用户请求的客户端不在我们已有的数据范围之内
 - 用户请求的资源过多，超过了我们已有的资源数量
 - 用户请求的资源超过了自己所需要的资源NEED。
 - 用户请求的资源假设分配之后，会带来死锁的问题。
- 为了能解决死锁的问题，也就是说，我们要模拟一个状态，这个状态是假设分配了这个资源，再来检查这个状态会不会死锁，如果在这种情况下，出现了死锁，那么我们不能分配，反正就可以分配
- 具体说来，检查这个是否会死锁的函数是 `int safety_algorithm(int *a)`，这个函数是核心。请看第四部分介绍。
- 当然，如果所有的检查都直接通过了，那么我们就会处理资源分配，具体说来
 - available的资源要减少，进行赋值修改。
 - need数组中，相关的需求也要减少。
 - allocation中，已经分配的资源要增加。

```

1 int requestBasicCheck(int *a{
2     int client_id = a[0];
```

```

3     if(client_id>=NUMBER_OF_CUSTOMERS)
4     {
5         printf("[Error] Request Error: The ID YOU INPUT Exceed THE
6             NUMBER_OF_CUSTOMERS\n");
7         return -1;
8     }
9     // basic check
10    for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
11    {
12        if(a[i]>need[client_id][i-1])
13        {
14            printf("[Error] Request Error: Your Request Exceed Need\n");
15            return -1;
16        }
17        if(a[i]>available[i-1])
18        {
19            printf("[Error] Request Error: Request Exceed Available\n");
20            return -1;
21        }
22    }
23    return 0;
24 /* request resource*/
25 int request(int *a)
26 {
27     int client_id = a[0];
28     if (requestBasicCheck(a) == -1)
29     {
30         return -1;
31     }
32     /* safety algorithm*/
33     int is_safe = safety_algorithm(a);
34     if(is_safe == -1)
35     {
36         printf("[Error] Your Request Leads to An Unsafe State! System
37 Denied!\n");
38         return 1;
39     }
40     else
41     {
42         printf("Request is satisfied.\n");
43     }
44     // after the request, the state should be safe.
45     for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
46     {
47         available[i-1] -= a[i];
48         need[client_id][i-1] -= a[i];
49         allocation[client_id][i-1] += a[i];
50     }

```

```
48 |     return 0;
49 }
```

0.1.3.4 4、核心算法二 银行家算法

- 首先我们来看教材的图片，我们严格遵守教材的算法过程。

8.6.3.1 Safety Algorithm

We can now present the algorithm for finding out whether or not a system is in a safe state. This algorithm can be described as follows:

- Let $Work$ and $Finish$ be vectors of length m and n , respectively. Initialize $Work = Available$ and $Finish[i] = false$ for $i = 0, 1, \dots, n - 1$.
- Find an index i such that both
 - $Finish[i] == false$
 - $Need_i \leq Work$If no such i exists, go to step 4.
- $Work = Work + Allocation_i$
 $Finish[i] = true$
Go to step 2.
- If $Finish[i] == true$ for all i , then the system is in a safe state.

This algorithm may require an order of $m \times n^2$ operations to determine whether a state is safe.

- 按照教材的过程，我们需要这么几个数组

- $Work$ 数组，这个数组初始化为 $Available$ 数组，这是因为直接操作 $Available$ 数组会很危险。
- 然后，在步骤2中，我们会不断寻找一个任务，这个任务满足两个条件，一是没有执行，也就是说 $Finish$ 的值是 $false$ ，同时，这个任务的需求值，在我们可以接收范围内。也就是说，这个任务请求的资源不会超过我们已有的资源池。
- 如果我们发现了一个任务，满足上面的这些条件，我们就可以直接执行了，执行后，需要对于 $Work$ 数组增加，因为执行后，已经被分配的资源都被释放了，也就是我们已有的资源（用 $Work$ 数组记录），同时也要给标记上已经完成。
- 这样执行之后，我们再去找下一轮可以执行的任务。直到找不到任务执行了。
- 这个时候就要检查，是否所有的任务都已经执行了，如果有任务没有执行，那就说明已经出现了死锁，反之，如果都执行了，这就说明这个状态不会出现死锁，是一个安全状态。
- 严格遵循我们的课本的逻辑，完成了下面代码：
 - 在完成的过程中，值得注意的是：我们是在模拟一些下一个步骤的状态，也就是说我们在检查下一个步骤会不会出现死锁，所以与课本的内容也不完全是一样的，要做一个适度的修改。
 - 预测下一个状态：`need[a[0]][j] (0 <= j < 资源数) 将会变成 need[a[0]][j] - a[j+1]`
`available[i] 会变成 available[i] - a[i+1];`
 - 由于 $need$ 数组针对不同的行，在未来可能的情况不一样，所以分别考虑。假如我们当前检查的行数，是我们要请求资源分配的行数， $need$ 数组未来会变成 $need[i][j] - a[j+1]$

```

2 int safety_algorithm(int *a)
3 {
4     // 首先，这一步我们要执行的是一个预测操作
5     // 假设这一个操作执行成功了，我们要检查是否会出现死锁
6     // 如果这个操作执行成功了，need[a[0]][j] (0<=j<资源数) 将会变成need[a[0]][j]-
7     a[j+1]
8     // 同样的，假如这个步骤执行成功了， available[i]会变成 available[i]-a[i+1];
9     // 你可能会问为什么是i+1，这是由于a[]系列的数组第一个a[0]元素是表示customer的编
10    号，所以有一个错位
11
12    // 所以按照教材，我们要定义一个数组，work数组最开始初始化为available数组，也就是假定
13    操作成功后的available
14    // 假如这个步骤执行成功了， available[i]会变成 available[i]-a[i+1];
15    int work[NODE_OF_RESOURCES];
16    for(int i=0; i<NUMBER_OF_RESOURCES; i++)
17        work[i] = available[i]-a[i+1];
18
19
20    // 然后我们要定义一个finish的数组，表示是否可以完成的任务，初始化为不能完成
21    int finish[NODE_OF_CUSTOMERS];
22    for(int i=0; i<NUMBER_OF_CUSTOMERS; ++i)
23        finish[i]=0;
24
25    int all_satisfy;
26    int i;
27    for(i=0; i<NUMBER_OF_CUSTOMERS; ++i)
28    {
29        all_satisfy=0; //refresh
30
31        if(finish[i]==0) {
32            for(int j=0; j<NUMBER_OF_RESOURCES; ++j)
33            {
34                // 由于need数组针对不同的行，在未来可能的情况不一样，所以分别考虑
35                // 假如我们当前检查的行数，是我们要请求资源分配的行数，need数组未来会变
36                成need[i][j]-a[j+1]
37                // 反之不会变的
38                if(i==a[0])
39                {
40                    if(need[i][j]-a[j+1]<=work[j])
41                        all_satisfy++;
42
43                    else if(need[i][j]<=work[j])
44                        all_satisfy++;
45                }
46
47                if(all_satisfy==NUMBER_OF_RESOURCES)
48                {
49                    //this customer can be finished
50                    for(int j=0; j<NUMBER_OF_RESOURCES; ++j)
51                        work[j] += allocation[i][j];
52                }
53            }
54        }
55    }
56}

```

```

45         finish[i]=1;
46         i=-1; // back to the start
47     }
48 }
49 }
50
51 int satisfy_number = 0;
52 for(i=0; i<NUMBER_OF_CUSTOMERS; ++i){
53     if(finish[i]==1)
54         satisfy_number++;
55     else{
56         printf("[Error] DeadLock Detected! If you do such a request, in
next status, Task %d won't be executed \n", i);
57     }
58 }
59 if(satisfy_number == NUMBER_OF_CUSTOMERS)
60     return 0; //safe
61 else
62     return -1; //unsafe
63 }
64

```

0.1.3.5 5、核心算法三 资源释放

- 对于用户请求的资源释放，我们也要类似资源请求的处理一样。
 - 首先我们要检查用户的请求参数是否合法，也要调用一个函数 `int releaseBasicCheck(int *a)` 检查是否可以执行。
 - 用户要释放的客户端不在我们已有的数据范围之内
 - 用户释放的资源过多，超过了已经请求的资源。
 - 当然，释放资源是不会带来死锁的，所以这一点不同于之前，可以不用考虑。

```

1
2 int releaseBasicCheck(int *a){
3     int consumer_id = a[0];
4     if(consumer_id >= NUMBER_OF_CUSTOMERS)
5     {
6         printf("[Error] Request Error: Your Request Release Exceed Need\n");
7         return -1;
8     }
9     // basic check
10    for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
11    {
12        if(a[i]>allocation[consumer_id][i-1])
13        {
14            printf("[Error] Release Error: Release Exceed Allocation\n");

```

```

15         return -1;
16     }
17 }
18 return 0;
19 }
20
21
22 /* release resource*/
23 int release(int *a)
24 {
25     int consumer_id = a[0];
26     if(releaseBasicCheck(a) == -1)
27         return -1;
28     // no exceed
29     for(int i=1;i<=NUMBER_OF_RESOURCES; ++i)
30     {
31         allocation[consumer_id][i-1] -= a[i];
32     }
33     return 0;
34 }
```

0.1.3.6 6、源代码展示

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #define NUMBER_OF_CUSTOMERS 5
5 #define NUMBER_OF_RESOURCES 4
6 #define SIZE 100
7 /* the available amount of each resource */
8 int available[NUMBER_OF_RESOURCES];
9 /*the maximum demand of each customer */
10 int maximum[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
11 /* the amount currently allocated to each customer */
12 int allocation[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
13 /* the remaining need of each customer */
14 int need[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];
15
16 int requestBasicCheck(int *a){
17     int client_id = a[0];
18     if(client_id>=NUMBER_OF_CUSTOMERS)
19     {
20         printf("[Error] Request Error: The ID YOU INPUT Exceed THE
21 NUMBER_OF_CUSTOMERS\n");
22         return -1;
23     }
24 }
```

```

22     }
23     // basic check
24     for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
25     {
26         if(a[i]>need[client_id][i-1])
27         {
28             printf("[Error] Request Error: Your Request Exceed Need\n");
29             return -1;
30         }
31         if(a[i]>available[i-1])
32         {
33             printf("[Error] Request Error: Request Exceed Available\n");
34             return -1;
35         }
36     }
37     return 0;
38 }
39
40
41 /* request resource*/
42 int request(int *a)
43 {
44     int client_id = a[0];
45     if (requestBasicCheck(a) == -1)
46         return -1;
47     /* safety algorithm*/
48     int is_safe = safety_algorithm(a);
49     if(is_safe == -1)
50     {
51         printf("[Error] Your Request Leads to An Unsafe State! System
52 Denied!\n");
53         return 1;
54     }
55     else
56     {
57         printf("Request is satisfied.\n");
58     }
59     // after the request, the state should be safe.
60     for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
61     {
62         available[i-1] -= a[i];
63         need[client_id][i-1] -= a[i];
64         allocation[client_id][i-1] += a[i];
65     }
66     return 0;
67 }
```

```

68 int releaseBasicCheck(int *a) {
69     int consumer_id = a[0];
70     if(consumer_id >= NUMBER_OF_CUSTOMERS)
71     {
72         printf("[Error] Request Error: Your Request Release Exceed
73 Need\n");
74     }
75     // basic check
76     for(int i=1;i<=NUMBER_OF_RESOURCES;++i)
77     {
78         if(a[i]>allocation[consumer_id][i-1])
79         {
80             printf("[Error] Release Error: Release Exceed Allocation\n");
81             return -1;
82         }
83     }
84     return 0;
85 }
86
87
88 /* release resource*/
89 int release(int *a)
90 {
91     int consumer_id = a[0];
92     if(releaseBasicCheck(a) == -1)
93     {
94         return -1;
95         // no exceed
96     }
97     for(int i=1;i<=NUMBER_OF_RESOURCES; ++i)
98     {
99         allocation[consumer_id][i-1] -= a[i];
100    }
101
102
103 /* show all the states*/
104 void state_display()
105 {
106     int i;
107     int j;
108     printf("_____ [Table Available
109 Resource] _____ \n");
110     for(i=0;i<NUMBER_OF_RESOURCES;++i)
111         printf("resource[%d]=%d ", i, available[i]);

```

```

111     printf("\n_____
");
112     printf("\n\n");
113
114     printf("_____ [Table
Maximum] _____ \n");
115     for(i=0;i<NUMBER_OF_CUSTOMERS;++i)
116     {
117         printf("custom[%d]    ", i);
118         for(j=0;j<NUMBER_OF_RESOURCES;++j)
119             printf("resource[%d]=%d ", j, maximum[i][j]);
120
121     printf("\n_____
");
122     }
123     printf("\n");
124
125     printf("_____ [Table
Allocation] _____ \n");
126     for(i=0;i<NUMBER_OF_CUSTOMERS;++i)
127     {
128         printf("custom[%d]    ", i);
129         for(j=0;j<NUMBER_OF_RESOURCES;++j)
130             printf("resource[%d]=%d ", j, allocation[i][j]);
131
132     printf("\n_____
");
133     }
134     printf("\n");
135
136     printf("_____ [Table
Need] _____ \n");
137     for(i=0;i<NUMBER_OF_CUSTOMERS;++i)
138     {
139         printf("custom[%d]    ", i);
140         for(j=0;j<NUMBER_OF_RESOURCES;++j)
141             printf("resource[%d]=%d ", j, need[i][j]);
142
143     printf("\n_____
");
144     }
145     printf("\n");

```

```

146     printf("_____[Table
147 End]_____\n");
148 }
149
150 int safety_algorithm(int *a)
151 {
152     // 首先，这一步我们要执行的是一个预测操作
153     // 假设这一个操作执行成功了，我们要检查是否会出现死锁
154     // 如果这个操作执行成功了，need[a[0]][j] (0<=j<资源数) 将会变成need[a[0]][j]-
155     // a[j+1]
156     // 同样的，假如这个步骤执行成功了，available[i]会变成 available[i]-a[i+1];
157     // 你可能会问为什么是i+1，这是由于a[]系列的数组第一个a[0]元素是表示customer的编
158     // 号，所以有一个错位
159
160     // 所以按照教材，我们要定义一个数组，work数组最开始初始化为available数组，也就是假
161     // 定操作成功后的available
162     // 假如这个步骤执行成功了，available[i]会变成 available[i]-a[i+1];
163     int work[NR_OF_RESOURCES];
164     for(int i=0; i<NR_OF_RESOURCES; i++)
165         work[i] = available[i]-a[i+1];
166
167     // 然后我们要定义一个finish的数组，表示是否可以完成的任务，初始化为不能完成
168     int finish[NR_OF_CUSTOMERS];
169     for(int i=0; i<NR_OF_CUSTOMERS; ++i)
170         finish[i]=0;
171
172     int all_satisfy;
173     int i;
174     for(i=0; i<NR_OF_CUSTOMERS; ++i)
175     {
176         all_satisfy=0; //refresh
177
178         if(finish[i]==0) {
179             for(int j=0; j<NR_OF_RESOURCES; ++j)
180             {
181                 // 由于need数组针对不同的行，在未来可能的情况不一样，所以分别考虑
182                 // 假如我们当前检查的行数，是我们要请求资源分配的行数，need数组未来会变
183                 // 成need[i][j]-a[j+1]
184                 // 反之不会变的
185                 if(i==a[0])
186                 {
187                     if(need[i][j]-a[j+1]<=work[j])
188                         all_satisfy++;
189                 }
190                 else if(need[i][j]<=work[j])
191                     all_satisfy++;
192             }
193         }
194     }
195 }
```

```

188     }
189     if(all_satisfy==NUMBER_OF_RESOURCES)
190     {   //this customer can be finished
191         for(int j=0; j<NUMBER_OF_RESOURCES; ++j)
192             work[j] += allocation[i][j];
193         finish[i]=1;
194         i=-1;// back to the start
195     }
196 }
197 }
198
199 int satisfy_number = 0;
200 for(i=0; i<NUMBER_OF_CUSTOMERS; ++i){
201     if(finish[i]==1)
202         satisfy_number++;
203     else{
204         printf("[Error] DeadLock Detected! If you do such a request, in
next status,Task %d won't be executed \n", i);
205     }
206 }
207 if(satisfy_number == NUMBER_OF_CUSTOMERS)
208     return 0;    //safe
209 else
210     return -1;  //unsafe
211 }
212
213
214 int main()
215 {
216     // initialize array MAXIMUM with txt file
217     FILE *in;
218     in = fopen("max_need.txt", "r");
219     char customer[SIZE];
220     char *temp;
221     for(int i=0;i<NUMBER_OF_CUSTOMERS;++i)
222     {
223         fgets(customer,SIZE,in);
224         temp = strdup(customer);
225         for(int j=0;j<NUMBER_OF_RESOURCES;j++)
226             maximum[i][j]=need[i][j]=atoi(strsep(&temp, ","));
allocation[i][j]=0;
227     }
228     fclose(in);
229
230     // initialize array **available** with txt file
231     in = fopen("available_resource.txt", "r");
232     char ava[SIZE];

```

```
233     fgets(ava, SIZE, in);
234     temp = strdup(ava);
235     for(int i=0;i<NUMBER_OF_RESOURCES;i++)
236         available[i]=atoi(strsep(&temp, ","));
237     fclose(in);
238
239
240     int input_instruction[NUMBER_OF_RESOURCES+1];
241     int request_value;
242     int release_value;
243     int tmp;
244
245     char input_line[SIZE];
246     int ifrun = 1;
247     while(ifrun)
248     {
249         printf("Banker >> ");
250         fgets(input_line, SIZE, stdin);
251
252         if(strcmp(input_line, "end\n") == 0)
253         {
254             ifrun = 0;
255             printf("[Info] Banker's Algorithm Finished, exit!\n");
256             continue;
257         }
258
259         if(input_line[0] == 'R' && input_line[1] == 'Q'){
260             //decode instruction
261             temp = strdup(input_line);
262             strsep(&temp, " ");
263             for(int i=0; i<=NUMBER_OF_RESOURCES; ++i)
264                 input_instruction[i] = atoi(strsep(&temp, " "));
265             request_value = request(input_instruction);
266             if(request_value!=0)
267                 printf("Request Failed. Please Follow Rules.\n");
268             continue;
269         }
270
271         else if(input_line[0] == 'R' && input_line[1] == 'L'){
272             temp = strdup(input_line);
273             strsep(&temp, " ");
274             for(int i=0; i<=NUMBER_OF_RESOURCES; ++i)
275                 input_instruction[i] = atoi(strsep(&temp, " "));
276             release_value = release(input_instruction);
277             if(release_value!=0)
278                 printf("Release Failed. Please Follow Rules.\n");
279             continue;

```

```

280     }
281
282     else if(input_line[0]=='*')
283         state_display();
284     else{
285         printf("[Error] Invalid Token, Please Input Again!\n");
286     }
287
288 }
289 return 0;
290 }
```

0.1.4 四、效果演示

- 能够打印出状态的表格。

```

zhangziqian@localhost:~/Des > end
[Info] Banker's Algorithm Finished, exit!
[zhangziqian@localhost Project6]$ ./banker
Banker >>> *
----- [Table Available Resource]
resource[0]=10 resource[1]=10 resource[2]=10 resource[3]=10
-----
----- [Table Maximum]
custom[0]   resource[0]=6 resource[1]=4 resource[2]=7 resource[3]=3
custom[1]   resource[0]=4 resource[1]=2 resource[2]=3 resource[3]=2
custom[2]   resource[0]=2 resource[1]=5 resource[2]=3 resource[3]=3
custom[3]   resource[0]=6 resource[1]=3 resource[2]=3 resource[3]=2
custom[4]   resource[0]=5 resource[1]=6 resource[2]=7 resource[3]=5
-----
----- [Table Allocation]
custom[0]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[1]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[2]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[3]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
custom[4]   resource[0]=0 resource[1]=0 resource[2]=0 resource[3]=0
-----
----- [Table Need]
custom[0]   resource[0]=6 resource[1]=4 resource[2]=7 resource[3]=3
custom[1]   resource[0]=4 resource[1]=2 resource[2]=3 resource[3]=2
custom[2]   resource[0]=2 resource[1]=5 resource[2]=3 resource[3]=3
custom[3]   resource[0]=6 resource[1]=3 resource[2]=3 resource[3]=2

```

The terminal window also displays a blurred background image of a video game scene from Genshin Impact, featuring a character and a landscape.

- 能够请求资源，并根据实际情况提示报错信息。

```
[zhangziqian@localhost:~/Des] + - X
-----
[Table Need]
custom[0]  resource[0]=6 resource[1]=4 resource[2]=7 resource[3]=3
custom[1]  resource[0]=4 resource[1]=2 resource[2]=3 resource[3]=2
custom[2]  resource[0]=2 resource[1]=5 resource[2]=3 resource[3]=3
custom[3]  resource[0]=6 resource[1]=3 resource[2]=3 resource[3]=2
custom[4]  resource[0]=5 resource[1]=6 resource[2]=7 resource[3]=5
-----
[Table End]
Banker >>> RQ 0 1 1 1 1
Request is satisfied.
Banker >>> RQ 1 2 2 2 2
Request is satisfied.
Banker >>> RQ 3 2 2 2 1
Request is satisfied.
Banker >>> RQ 2 1 3 2 2
Request is satisfied.
Banker >>> RQ 4 1 1 1 1
Request is satisfied.
Banker >>> RQ 3 1 1 1 1
Request is satisfied.
Banker >>> RQ 1 2 2 2 2
[Error] Request Error: Your Request Exceed Need
Request Failed. Please Follow Rules.
Banker >>> RQ 2 1 1 1 1
[Error] Request Error: Request Exceed Available
Request Failed. Please Follow Rules.
Banker >>> *
-----
[Table Available Resource]
resource[0]=2 resource[1]=0 resource[2]=1 resource[3]=2
-----
[Table Maximum]
custom[0]  resource[0]=6 resource[1]=4 resource[2]=7 resource[3]=3
```

25°C



13:57 2022/5/7

