操作系统 project-6 实验报告

* 姓名: 管仁阳　学号:519021911058　邮箱: guanrenyang@sjtu.edu.cn

1 实验名称

Banker’s Algorithm

2 实验任务

编写 C 程序来实现银行家算法。

1. **客户**从**银行**中请求或释放资源。
2. **银行家**当且仅当此次申请使得系统保持**安全状态**时满足申请。
3. 当此次请求使得系统处于不安全状态时银行家拒绝请求。

3 预备知识

3.1 预定义的数据结构

3.1.1 Available

一个长度为 m 的向量代表每种资源种类的可获得实例数。$Allocation[i]$ 代表第 i 种资源的可获得实例数。

3.1.2 Max

一个 $n \times m$ 矩阵代表每个线程所需要的每种资源的最大数量。$Max[i][j]$ 代表线程 $Thread_i$ 所需要的资源种类 $Resource_j$ 的最大数量。

3.1.3 Allocation

一个 $n \times m$ 矩阵代表每个线程所被分配的每种资源的数量。$Allocation[i][j]$ 代表线程 $Thread_i$ 被分配到的资源种类 $Resource_j$ 的数量。

3.1.4 Need

一个 $n \times m$ 矩阵代表每个线程所还需要的每种资源的数量。$Need[i][j]$ 代表线程 $Thread_i$ 还需要的的资源种类 $Resource_j$ 的数量。

3.2 银行家算法

3.2.1 算法描述

当新线程进入系统时，它必须声明它可能需要的每个资源类型的最大实例个数，它可能不是超过系统中的资源总数。当用户请求设置资源时，系统必须确定分配这些资源是否将使系统处于安全状态。如果此次分配让系统处于安全状态则分配资源；否则，线程必须等待其他线程释放足够的内存资源。**伪代码如下所示**

1. 定义长度为 m 的数组 $Work$ 和长度为 n 的数组 $Finish$。初始化为 $Work = Available$ 和 $Finish[i] = false$, $i = 0, 1, \cdots, n-1$

2. 找一个下标 i 满足：

   (a) $Finish[i] == false$

   (b) $Need_i \leq Work$

   如果没有这样的 i 请跳到步骤 4。

3. (a) $Work = Work + Allocation_i$

   (b) $Finish[i] = true$

   跳转到步骤 2.

4. 如果对所有 i 都满足 $Finish[i] == true$，就代表系统处于安全状态。

3.2.2 资源分配算法

定义长度为 m 的数组 $Request_i$ 代表某一线程请求的各种资源的数量。

1. 如果 $Request_i \leq Need_i$，跳转步骤 2。否则请求失败。

2. 如果 $Request_i \leq Allocation_i$，跳转步骤 3。否则请求失败。

3. (a) $Available = Available–Request_i$

   (b) $Allocation_i = Allocation_i + Request_i$

   (c) $Need_i = Need_i–Request_i$

   执行以上三部模拟分配以后的状态。如果此状态是安全状态，则通过请求。

4 实验内容

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#define NUMBER_OF_CUSTOMERS 5
#define NUMBER_OF_RESOURCES 4
#define SUCCESS 0
/*the available amount of each resource*/
int available[NUMBER_OF_RESOURCES]; //available[NUMBER_OF_RESOURCES]

/*the maximum demand of each cuotomer*/
int maximum[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];

/*the amount currently allocated to each customer*/
int allocation[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];

/*the remaining need of each customer*/
int need[NUMBER_OF_CUSTOMERS][NUMBER_OF_RESOURCES];

```

```
20 int requset_resources(int customer_num,int requset[]);
21 void release_resources(int customer_num,int release[]);
22
23 void display();
24 //tool functions
25 int is_available(int requset[]);
26 int is_need(int customer_num,int request[]);
27 int satisfy_one_task(int Work[],int Finish[]);
28 int main(int argc,char* argv[])
29 {
30         if(argc!=5)
31         {
32             fprintf(stderr,"ERROR");
33             exit(1);
34         }
35     available[0]=atoi(argv[1]);
36     available[1]=atoi(argv[2]);
37     available[2]=atoi(argv[3]);
38     available[3]=atoi(argv[4]);
39     FILE* in;
40     in=fopen("maximum.txt","r");
41         if(in==NULL)
42         {
43             fprintf(stderr,"ERROR");
44             exit(1);
45         }
46
47     int t=0;
48     char* maximum_i=(char*) malloc(10*sizeof(char));
49     while(fgets(maximum_i,10,in)!=NULL)
50     {
51         if(maximum_i[0]>'9'||maximum_i[0]<'1')
52             break;
53         char * temp=strdup(maximum_i);
54         for(int i=0;i<NUMBER_OF_RESOURCES;++i)
55         {
56             maximum[t][i]=atoi(strsep(&temp,","));
57             need[t][i]=maximum[t][i];
58         }
59         t++;
60     }
61     free(maximum_i);
62     char* user_command=(char*)malloc(20*sizeof(char));
63     while(fgets(user_command,20,stdin))
64     {
65         if(user_command[0]=='*')
66         {
67             display();
68             continue;
69         }
```

```
70          char * temp=strdup(user_command);
71
72          int temp_resources[NUMBER_OF_RESOURCES];
73
74          char * command_name=strsep(&temp," ");
75          int customer_num=atoi(strsep(&temp," "));
76          for(int i=0;i<NUMBER_OF_RESOURCES;++i)
77          {
78              char * para=strsep(&temp," ");
79              if(para==NULL)
80              {
81                  fprintf(stderr,"Synax Error: Missing parameters\n");
82                  exit(1);
83              }
84
85              temp_resources[i]=atoi(para);
86          }
87          if(strcmp(command_name,"RQ")==0)//RQ command
88          {
89              int succ=requset_resources(customer_num,temp_resources);
90
91              if(succ==SUCCESS)
92                  fprintf(stdout,"Satisfied\n");
93              else
94                  fprintf(stdout,"Denied\n");
95          }
96          else if(strcmp(command_name,"RL")==0)//RL command
97              release_resources(customer_num,temp_resources);
98          else
99          {
100             fprintf(stderr,"Synax Error: Wrong command name\n");
101             continue;
102         }
103     }
104     return 0;
105 }
106 int requset_resources(int customer_num,int request[]){
107     if(is_available(request)==!SUCCESS)
108         return -1;
109     if(is_need(customer_num,request)==!SUCCESS)
110         return -1;
111     for(int i=0;i<NUMBER_OF_RESOURCES;++i)
112     {
113         available[i]-=request[i];
114         allocation[customer_num][i]+=request[i];
115         need[customer_num][i]-=request[i];
116     }
117     //start banker's algorithm
118     int Work[NUMBER_OF_RESOURCES];
119     for(int i=0;i<NUMBER_OF_RESOURCES;++i)
```

```
120         Work[i]=available[i];
121
122     int Finish[NUMBER_OF_CUSTOMERS];
123     for(int i=0;i<NUMBER_OF_CUSTOMERS;++i)
124         Finish[i]=0;
125
126     for(int i=0;i<NUMBER_OF_CUSTOMERS;++i)
127     {
128         int next_satisfied=satisfy_one_task(Work,Finish);
129
130         if(next_satisfied==-1)
131             return -1;
132
133         Finish[next_satisfied]=1;
134
135         for(int j=0;j<NUMBER_OF_RESOURCES;++j)
136             Work[j]+=allocation[next_satisfied][j];
137     }
138
139     return SUCCESS;
140 }
141 int is_available(int request[]){
142     for(int i=0;i<NUMBER_OF_RESOURCES;++i)
143     {
144         if(request[i]>available[i])
145             return -1;
146     }
147     return SUCCESS;
148 }
149 int is_need(int customer_num,int request[]){
150     for(int i=0;i<NUMBER_OF_RESOURCES;++i)
151     {
152         if(request[i]>need[customer_num][i])
153             return -1;
154     }
155     return SUCCESS;
156 }
157 int satisfy_one_task(int Work[],int Finish[]){
158     for(int i=0;i<NUMBER_OF_CUSTOMERS;++i)
159     {
160         if(Finish[i]!=0)
161             continue;
162         int flag=0;
163         for(int j=0;j<NUMBER_OF_RESOURCES;++j)
164         {
165             if(need[i][j]>Work[j])
166             {
167                 flag=1;
168                 break;
169             }
```

```
170         }
171         if(flag==0)
172             return i;
173     }
174     return -1;
175 }
176 void release_resources(int customer_num,int release[]){
177     for(int i=0;i<NUMBER_OF_RESOURCES;++i)
178     {
179         if(allocation[customer_num][i]<release[i])
180         {
181             fprintf(stdout,"Customer %d has only %d instances of resource typ
182             available[i]+=allocation[customer_num][i];
183             allocation[customer_num][i]=0;
184         }
185         else
186         {
187             available[i]+=release[i];
188             allocation[customer_num][i]-=release[i];
189         }
190     }
191 }
192 void display(){
193     //display available
194     fprintf(stdout,"Available:\n");
195     for(int i=0;i<NUMBER_OF_RESOURCES;++i)
196         fprintf(stdout,"%d, ",available[i]);
197     fprintf(stdout,"\n");
198 
199     //display Maximum
200     fprintf(stdout,"Maximum:\n");
201     for(int i=0;i<NUMBER_OF_CUSTOMERS;++i)
202     {
203         for(int j=0;j<NUMBER_OF_RESOURCES;++j)
204         {
205             fprintf(stdout,"%d, ",maximum[i][j]);
206         }
207         fprintf(stdout,"\n\n");
208     }
209 
210     //display Allocation
211     fprintf(stdout,"Allocation:\n");
212     for(int i=0;i<NUMBER_OF_CUSTOMERS;++i)
213     {
214         for(int j=0;j<NUMBER_OF_RESOURCES;++j)
215         {
216             fprintf(stdout,"%d, ",allocation[i][j]);
217         }
218         fprintf(stdout,"\n\n");
219     }
```

```
220     //display need
221     fprintf(stdout,"Need:\n");
222     for(int i=0;i<NUMBER_OF_CUSTOMERS;++i)
223     {
224         for(int j=0;j<NUMBER_OF_RESOURCES;++j)
225         {
226             fprintf(stdout,"%d, ",need[i][j]);
227         }
228         fprintf(stdout,"\n\n");
229     }
230 }
```

5 实验结果

模拟结果与性能指标见图 1.

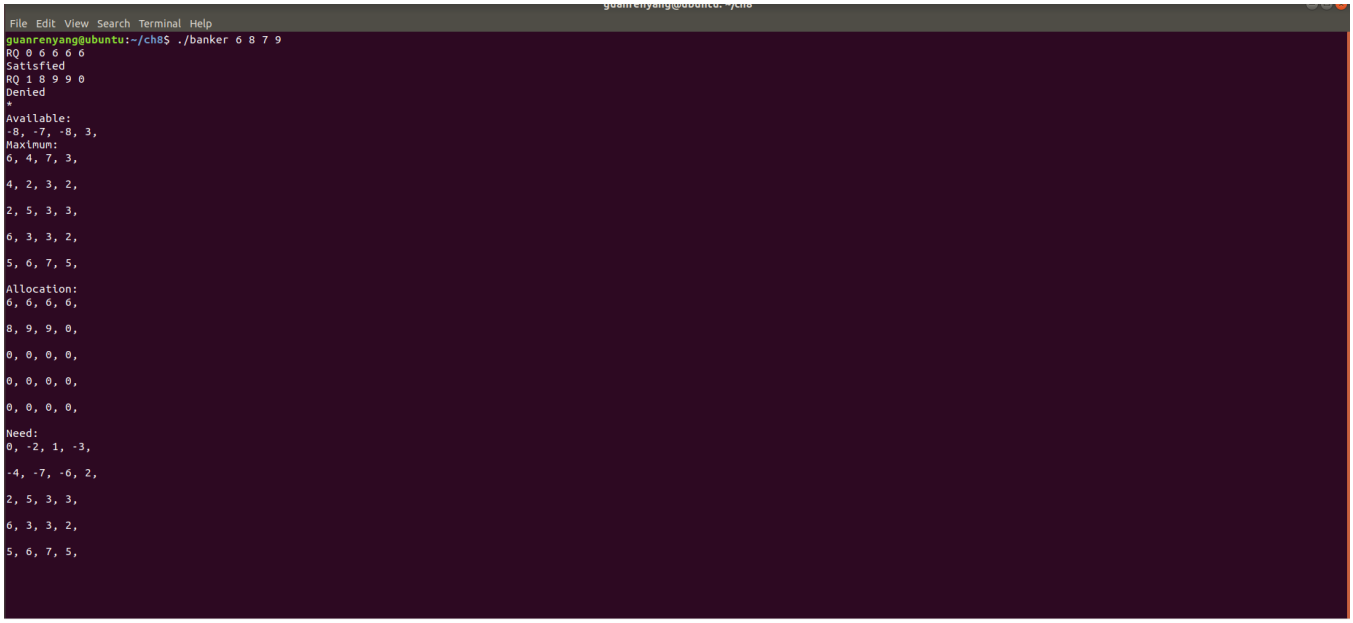

图 1: Result

6 总结与思考

通过此次实验我编写程序模拟了银行家算法。更加深入地理解了银行家算法的实现步骤、安全状态的判别算法与资源分配算法。银行家算法是多线程资源分配的有效方式，这次促使我认识到并行编程的重要特性。