

01 实验概述

1.1 实验目的

银行家算法是操作系统中避免死锁的典型算法，通过本实验加深对银行家算法的理解。

1.2 实验内容

用C语言或C++编写一个简单的银行家算法模拟程序，实现多个进程争用系统临界资源时的分配过程。要求实现如下功能：

1. 初始化当前系统状态。
2. 显示当前资源分配情况。
3. 检查当前系统是否安全。
4. 分配资源。某进程请求分配一组资源时，先进行安全性检查，分配后系统是否处于安全状态？如果安全，显示安全序列，将资源分配给该进程，否则该进程等待。
5. 新建一个进程。
6. 撤销一个进程。

02 实验设计与实现

2.1 银行家算法所需数据结构

假设实验共有 m 个资源类别， n 个进程。

银行家算法的数据结构主要有以下几个：

1. Available[n] :
 - 表示系统中可用的资源数目;
 - Available[j] 表示第j类资源的可用数目;
2. Max[n][m] :
 - 表示各进程对各类资源的最大需求量;
 - Max[i][j] 表示进程i对第j类资源的最大需求量;
3. Allocation[n][m] :
 - 表示系统已分配给各进程的各类资源数目;
 - Allocation[i][j] 表示进程i已分配给第j类资源的数目;
4. Need[n][m] :
 - 表示各进程对各类资源的尚需求量;
 - Need[i][j] 表示进程i对第j类资源的尚需求量;
 - $Need[i][j] = Max[i][j] - Allocation[i][j]$;
5. Request[m] :
 - 表示进程一次请求对各类资源的请求量;
 - Request[j] 表示进程请求第j类资源的数目;
6. Work[m] :
 - 表示系统可提供给进程继续运行所需的各类资源数目;
 - 初始值为 Available[m] ;

对于上述数据结构，在实现上使用C++的vector容器进行封装，以便于使用。

定义如下全局变量：

```
1  vector<string> resource_name;    // 资源名称
2  vector<int> Available;           // 可用资源个数
3  vector<vector<int>> Max;          // 最大需求矩阵
4  vector<vector<int>> Allocation;  // 已分配矩阵
5  vector<vector<int>> Need;         // 需求矩阵
6  int resource_num;               // 资源种类数
7  int task_num;                   // 进程数
8
```

2.2 数据结构封装

为方便实验，下面对vector的部分运算符进行重载。

2.2.1 重载运算符 <<

重载运算符 <<，使得可以直接使用 cout 输出vector容器的内容。

```
1 // 重载vector输出
2 template <typename T>
3 std::ostream &operator<<(std::ostream &o, const std::vector<T> &vec) {
4     int num = vec.size();
5     for (int i = 0; i < num; i++)
6         cout << (i ? " " : "") << vec[i];
7     return o;
8 }
```

2.2.2 重载运算符 +

重载运算符 +，使得可以直接使用 vector 容器进行向量加法运算。

```
1 // 重载vector加法
2 vector<int> operator+(vector<int> &a, vector<int> &b) {
3     if (a.size() != b.size()) {
4         cout << "vector运算的两个操作数大小不同! \n";
5         exit(-1);
6     }
7     int n = a.size();
8     vector<int> res(n);
9     for (int i = 0; i < n; i++) {
10         res[i] = a[i] + b[i];
11     }
12     return res;
13 }
```

2.2.3 重载运算符 -

重载运算符 - ，使得可以直接使用 vector 容器进行向量减法运算。

```
1 // 重载vector减法
2 vector<int> operator-(vector<int> &a, vector<int> &b) {
3     if (a.size() != b.size()) {
4         cout << "vector运算的两个操作数大小不同! \n";
5         exit(-1);
6     }
7     int n = a.size();
8     vector<int> res(n);
9     for (int i = 0; i < n; i++) {
10         res[i] = a[i] - b[i];
11     }
12     return res;
13 }
```

2.2.4 重载运算符 <=

重载运算符 <= ，使得可以直接使用 vector 容器进行向量比较运算（用于判断请求量与现有资源的大小关系）。

```
1 // 重载vector比较<=, 用于判断是否资源数小于等于关系
2 bool operator<=(vector<int> const &a, vector<int> const &b) {
3     if (a.size() != b.size()) {
4         cout << "vector运算的两个操作数大小不同! \n";
5         exit(-1);
6     }
7     int n = a.size();
8     for (int i = 0; i < n; i++) {
9         if (a[i] > b[i]) return false;
10    }
11    return true;
12 }
```

2.3 初始化系统状态

读入资源种类数 `resource_num` 和进程数 `task_num`，并初始化 `Available`、`Max`、`Allocation`、`Need` 等数据结构。

```
1 // 初始化，获取用户输入，初始化各个变量(资源数，进程数，各个矩阵)
2 void Init() {
3     string name; int num;
4     cout << "*****单处理机系统进程调度实现*****" << '\n';
5     cout << "请首先输入系统可供资源种类的数量:";
6     cin >> resource_num;
7     resource_name.resize(resource_num);
8     Available.resize(resource_num);
9     for (int i = 0; i < resource_num; i++) {
10         cout << "资源" << i+1 << "的名称:";
11         cin >> resource_name[i];
12         cout << "资源的数量:";
13         cin >> Available[i];
14     }
15     cout << '\n';
16     cout << "请输入作业的数量:";
17     cin >> task_num;
18     Max = Allocation = Need =
19         vector<vector<int>>(task_num, vector<int>(resource_num, 0));
20
21     printf("请输入各进程的最大需求量(%d*d矩阵)[Max]:\n", task_num, resource_num);
22     for (int i = 0; i < task_num; i++)
23         for (int j = 0; j < resource_num; j++)
24             cin >> Max[i][j];
25
26     printf("请输入各进程已经申请的资源量(%d*d矩阵)[Allocation]:\n", task_num,
27 resource_num);
28     for (int i = 0; i < task_num; i++)
29         for (int j = 0; j < resource_num; j++)
30             cin >> Allocation[i][j];
31     GetNeed(); // 由Max和Allocation计算Need
32     cout << '\n';
33     ListResourceStatus(); // 显示当前资源分配情况
34     CheckStatusSecurity(); // 检查当前状态安全性
35     cout << "\n\n";
36 }
```

2.4 显示当前资源分配情况(操作1)

显示当前资源分配情况，包括 Available、Max、Allocation、Need 等数据结构。

```
1 // 显示当前资源分配情况
2 // 操作1, 显示当前资源分配情况
3 void ListResourceStatus() {
4     cout << "系统目前可用的资源[Avaliable]: " << '\n';
5     cout << resource_name << '\n';
6     cout << Available << '\n';
7
8     cout << "          Max      Allocation      Need" << '\n';
9     cout << "进程名      ";
10    for (int i = 0; i < 3; i++)
11        cout << resource_name << "          ";
12    cout << '\n';
13
14    for (int i = 0; i < task_num; i++) {
15        cout << " " << i << "          ";
16        cout << Max[i] << "          ";
17        cout << Allocation[i] << "          ";
18        cout << Need[i] << '\n';
19    }
20 }
```

2.5 检查当前系统是否安全(操作2)

检查当前系统是否安全，如果安全，显示安全序列，否则显示不安全。

算法如下：

1. 令 $Work = Available$ ；令 $Finish[i] = false$ ，表示进程 i 未完成；
2. 从进程集合中找到一个满足以下条件的进程 i ：
 - $Finish[i] == false$ ；
 - $Need[i] \leq Work$ ；
3. 找到这样的进程 i ，则执行以下操作：
 1. $Work = Work + Allocation[i]$ ；

2. Finish[i] = true ;
3. 转到步骤3;
4. 如果找不到这样的进程i, 则转到步骤4;
4. 如果所有进程都满足 Finish[i] == true , 则系统是安全的, 且安全序列为 Finish 中为true的进程序列; 否则系统是不安全的。

```
1 // 操作2, 检查当前状态安全性, 若安全则输出安全序列; 否则输出不安全
2 // 返回值: 是否安全
3 bool CheckStatusSecurity() {
4     vector<int> Work = Available;
5     vector<bool> Finish(task_num, false);
6     vector<int> SafeSequence;
7     for (int count = 0; count < task_num; count++) {
8         bool flag = false;
9         for (int i = 0; i < task_num; i++) {
10             if (Finish[i] == false && Need[i] <= Work) {
11                 Work = Work + Allocation[i];
12                 Finish[i] = true;
13                 SafeSequence.push_back(i);
14                 flag = true;
15                 break;
16             }
17         }
18         if (flag == false) {
19             cout << "系统处于不安全状态, 无法分配! " << '\n';
20             return false;
21         }
22     }
23     cout << "系统是安全的!" << '\n';
24     cout << "安全序列为: ";
25     for (int i = 0; i < task_num; i++)
26         cout << SafeSequence[i] << (i == task_num - 1 ? "" : "->");
27     cout << '\n';
28     return true;
29 }
```

2.6 分配资源(操作3)

读入进程号 `task_id` 和请求量 `Request`，判断是否安全，如果安全，显示安全序列，将资源分配给该进程，否则该进程等待。

判断是否安全的算法如下：

1. 判断 `Request <= Need[task_id]`；
2. 判断 `Request <= Available`；
3. 尝试将资源分配给进程 `task_id`，计算分配后状态：
 - `Available = Available - Request`；
 - `Allocation[task_id] = Allocation[task_id] + Request`；
 - `Need[task_id] = Need[task_id] - Request`；
4. 判断分配后的状态是否安全
 - 这里可以直接调用操作2的函数；
5. 根据判断结果，执行以下操作：
 - 如果安全，显示安全序列，将资源分配给该进程；
 - 否则，回滚分配，不进行此次分配。

```
1 // 操作3，分配资源
2 // 先判断是否安全，若安全则分配，否则不分配（回滚）
3 void AllocateResource() {
4     int task_id;
5     cout << "请输入要求分配的资源进程号0-" << task_num - 1 << ":";
6     cin >> task_id;
7     vector<int> Request(resource_num);
8     cout << "请输入进程 " << task_id << " 申请的资源:\n";
9     for (int i = 0; i < resource_num; i++)
10    {
11        cout << resource_name[i] << ":";
12        cin >> Request[i];
13    }
14    if (Request > Need[task_id]) {
15        cout << "进程" << task_id << "申请的资源大于其需要量 分配出错，不予分配!"
16            << '\n';
17        return;
18    }
19    if (Request > Available) {
```



```

20         cout << "进程" << task_id << "申请的资源大于系统现在可利用的资源 分配出错，不予分
    配!"
21         << '\n';
22         return;
23     }
24     Available = Available - Request;
25     Allocation[task_id] = Allocation[task_id] + Request;
26     Need[task_id] = Need[task_id] - Request;
27
28     cout << '\n';
29     ListResourceStatus();
30     if (CheckStatusSecurity() == false)
31     {
32         // 回滚
33         Available = Available + Request;
34         Allocation[task_id] = Allocation[task_id] - Request;
35         Need[task_id] = **** Need[task_id] + Request;
36
37         cout << "系统目前可用的资源[Avaliable]仍然为:" << '\n';
38         cout << resource_name << '\n';
39         cout << Available << '\n';
40     }
41 }

```

2.7 新建进程(操作4)

新建进程，读入新进程的 Max 和 Allocation，并计算 Need，将信息添加到全局 Max、Allocation、Need 中。

```

1 // 操作4，创建进程
2 void CreateProcess() {
3     // 获取新进程信息
4     vector<int> process_max(resource_num);
5     cout << "请输入进程最大需求量:\n";
6     for (int i = 0; i < resource_num; i++) {
7         cout << resource_name[i] << ":";
8         cin >> process_max[i];
9     }
10
11     vector<int> process_allocation(resource_num);
12     cout << "请输入进程已经申请的资源量:\n";
13     for (int i = 0; i < resource_num; i++) {

```

```

14         cout << resource_name[i] << ":";
15         cin >> process_allocation[i];
16     }
17
18     vector<int> process_need = process_max - process_allocation;
19
20     // 更新各个矩阵
21     Max.push_back(process_max);
22     Allocation.push_back(process_allocation);
23     Need.push_back(process_need);
24
25     // 更新进程数
26     task_num++;
27
28     // 显示当前进程及资源分配情况
29     cout << "进程创建成功!" << '\n';
30     cout << "当前进程及资源分配情况:" << '\n';
31     ListResourceStatus();
32 }

```

2.8 撤销进程(操作5)

撤销进程，读入进程号 `task_id`，将该进程的 `Max`、`Allocation`、`Need` 信息从全局变量中删除，同时回收该进程的资源到 `Available`。

```

1 // 操作5，删除进程
2 void DeleteProcess() {
3     // 获取要删除的进程号
4     int process_id;
5     cout << "请输入要删除的进程号0-" << task_num - 1 << ":";
6     cin >> process_id;
7
8     // 更新各个矩阵
9     Available = Available + Allocation[process_id];
10    Max.erase(Max.begin() + process_id);
11    Allocation.erase(Allocation.begin() + process_id);
12    Need.erase(Need.begin() + process_id);
13
14    // 更新进程数
15    task_num--;
16
17    // 显示当前进程及资源分配情况

```

```

18     cout << "进程删除成功!" << '\n';
19     cout << "当前进程及资源分配情况:" << '\n';
20     ListResourceStatus();
21 }

```

2.9 操作选择与主函数

根据用户输入的操作号，执行相应的操作(调用上述操作函数)。

```

1 // 获取用户输入的选项
2 int GetOption() {
3     cout << "*****银行家算法演示*****" << '\n';
4     cout << "1:显示当前资源分配情况" << '\n';
5     cout << "2:检查当前状态安全性" << '\n';
6     cout << "3:分配资源" << '\n';
7     cout << "4:创建进程" << '\n';
8     cout << "5:删除进程" << '\n';
9     cout << "6:离开" << '\n';
10    cout << "*****" << '\n';
11    cout << "请选择功能号: ";
12    int op;
13    cin >> op;
14    return op;
15 }
16 int main() {
17     Init();
18     int op;
19     while (op = GetOption()) {
20         switch (op) {
21             case 1:
22                 ListResourceStatus(); // 显示当前资源分配情况
23                 break;
24             case 2:
25                 CheckStatusSecurity(); // 检查当前状态安全性
26                 break;
27             case 3:
28                 AllocateResource(); // 分配资源
29                 break;
30             case 4:
31                 CreateProcess(); // 创建进程
32                 break;
33             case 5:

```

```

34         DeleteProcess(); // 删除进程
35         break;
36     case 6:
37         return 0; // 退出
38     default:
39         cout << "输入错误, 请重新输入!" << '\n';
40     }
41     cout << '\n' << '\n';
42 }
43 return 0;
44 }

```

03 实验测试与结果

使用 银行家测试数据例子.txt 数据，对操作1~3进行测试，同时自选数据测试了操作4、5（创建进程和删除进程）功能。

测试结果如下：

（注：由于测试数据较多，这里略去多余空行及部分操作选择提示）

```

1  PS D:\files\Course\2023Fal-操作系统\lab\lab5> g++ .\banker.cpp
2  PS D:\files\Course\2023Fal-操作系统\lab\lab5> .\a.exe
3  *****单处理机系统进程调度实现*****
4  请首先输入系统可供资源种类的数量:3
5  资源1的名称:a
6  资源的数量:3
7  资源2的名称:b
8  资源的数量:3
9  资源3的名称:c
10 资源的数量:2
11 请输入作业的数量:5
12 请输入各进程的最大需求量(5*3矩阵)[Max]:
13 7 5 3
14 3 2 2
15 9 0 2
16 2 2 2
17 4 3 3
18 请输入各进程已经申请的资源量(5*3矩阵)[Allocation]:
19 0 1 0
20 2 0 0

```

```

21      3 0 2
22      2 1 1
23      0 0 2
24      系统目前可用的资源[Avaliable]:
25      a b c
26      3 3 2
27
28      Max      Allocation      Need
29      进程名   a b c      a b c      a b c
30      0        7 5 3      0 1 0      7 4 3
31      1        3 2 2      2 0 0      1 2 2
32      2        9 0 2      3 0 2      6 0 0
33      3        2 2 2      2 1 1      0 1 1
34      4        4 3 3      0 0 2      4 3 1
35
36      系统是安全的!
37      安全序列为: 1->3->0->2->4
38
39      *****银行家算法演示*****
40
41      *          1:显示当前资源分配情况
42      *          2:检查当前状态安全性
43      *          3:分配资源
44      *          4:创建进程
45      *          5:删除进程
46      *          6:离开
47
48      *****
49
50      请选择功能号: 1
51
52      系统目前可用的资源[Avaliable]:
53      a b c
54      3 3 2
55
56      Max      Allocation      Need
57      进程名   a b c      a b c      a b c
58      0        7 5 3      0 1 0      7 4 3
59      1        3 2 2      2 0 0      1 2 2
60      2        9 0 2      3 0 2      6 0 0
61      3        2 2 2      2 1 1      0 1 1
62      4        4 3 3      0 0 2      4 3 1
63
64      *****银行家算法演示*****
65
66      请选择功能号: 2
67
68      系统是安全的!
69      安全序列为: 1->3->0->2->4
70
71      *****银行家算法演示*****
72
73      请选择功能号: 3
74
75      请输入要求分配的资源进程号0-4:1
76
77      请输入进程 1 申请的资源:
78
79      a:1
80
81      b:0
82
83      c:2

```

```

66 系统目前可用的资源[Avaliable]:
67 a b c
68 2 3 0
69
70      Max      Allocation      Need
71 进程名    a b c      a b c      a b c
72 0         7 5 3      0 1 0      7 4 3
73 1         3 2 2      3 0 2      0 2 0
74 2         9 0 2      3 0 2      6 0 0
75 3         2 2 2      2 1 1      0 1 1
76 4         4 3 3      0 0 2      4 3 1
77
78 系统是安全的!
79 安全序列为: 1->3->0->2->4
80
81 *****银行家算法演示*****
82 请选择功能号: 3
83 请输入要求分配的资源进程号0-4:4
84 请输入进程 4 申请的资源:
85 a:3
86 b:3
87 c:0
88 进程4申请的资源大于系统现在可利用的资源 分配出错, 不予分配!
89 *****银行家算法演示*****
90 请选择功能号: 3
91 请输入要求分配的资源进程号0-4:0
92 请输入进程 0 申请的资源:
93 a:0
94 b:2
95 c:0
96 系统目前可用的资源[Avaliable]:
97 a b c
98 2 1 0
99
100      Max      Allocation      Need
101 进程名    a b c      a b c      a b c
102 0         7 5 3      0 3 0      7 2 3
103 1         3 2 2      3 0 2      0 2 0
104 2         9 0 2      3 0 2      6 0 0
105 3         2 2 2      2 1 1      0 1 1
106 4         4 3 3      0 0 2      4 3 1
107
108 系统处于不安全状态, 无法分配!
109 系统目前可用的资源[Avaliable]仍然为:
110 a b c
111 2 3 0
112
113 *****银行家算法演示*****
114 请选择功能号: 2
115 系统是安全的!
116 安全序列为: 1->3->0->2->4

```

```

111 *****银行家算法演示*****
112 请选择功能号: 1
113 系统目前可用的资源[Avaliable]:
114 a b c
115 2 3 0
116
117      Max      Allocation      Need
118 进程名  a b c      a b c      a b c
119 0      7 5 3      0 1 0      7 4 3
120 1      3 2 2      3 0 2      0 2 0
121 2      9 0 2      3 0 2      6 0 0
122 3      2 2 2      2 1 1      0 1 1
123 4      4 3 3      0 0 2      4 3 1
124 *****银行家算法演示*****
125 请选择功能号: 5
126 请输入要删除的进程号0-4:4
127 进程删除成功!
128 当前进程及资源分配情况:
129 系统目前可用的资源[Avaliable]:
130 a b c
131 2 3 2
132
133      Max      Allocation      Need
134 进程名  a b c      a b c      a b c
135 0      7 5 3      0 1 0      7 4 3
136 1      3 2 2      3 0 2      0 2 0
137 2      9 0 2      3 0 2      6 0 0
138 3      2 2 2      2 1 1      0 1 1
139 *****银行家算法演示*****
140 请选择功能号: 5
141 请输入要删除的进程号0-3:0
142 进程删除成功!
143 当前进程及资源分配情况:
144 系统目前可用的资源[Avaliable]:
145 a b c
146 2 4 2
147
148      Max      Allocation      Need
149 进程名  a b c      a b c      a b c
150 0      3 2 2      3 0 2      0 2 0
151 1      9 0 2      3 0 2      6 0 0
152 2      2 2 2      2 1 1      0 1 1
153 *****银行家算法演示*****
154 请选择功能号: 2
155 系统是安全的!
156 安全序列为: 0->2->1
157 *****银行家算法演示*****
158 请选择功能号: 4

```

```

156  请输入进程最大需求量:
157  a:3
158  b:3
159  c:3
160  请输入进程已经申请的资源量:
161  a:1
162  b:1
163  c:1
164  进程创建成功!
165  当前进程及资源分配情况:
166  系统目前可用的资源[Avaliable]:
167  a b c
168  2 4 2
169
170      Max      Allocation      Need
171  进程名    a b c      a b c      a b c
172  0         3 2 2      3 0 2      0 2 0
173  1         9 0 2      3 0 2      6 0 0
174  2         2 2 2      2 1 1      0 1 1
175  3         3 3 3      1 1 1      2 2 2
176
177  *****银行家算法演示*****
178  请选择功能号: 2
179  系统是安全的!
180  安全序列为: 0->2->1->3
181  *****银行家算法演示*****
182  请选择功能号: 6
183
184  PS D:\files\Course\2023Fal-操作系统\lab\lab5>

```

04 实验总结

本次实验主要是对银行家算法的实现。银行家算法是一种避免死锁的算法，通过在系统的资源分配前进行安全性检查，从而避免死锁的发生。而在安全性检查中，重要的如何是判断系统是否处于安全状态，即能否找到一个安全路径，使得所有进程能够完成任务。

对于本次实验的实现，通过C++的vector容器对银行家算法的数据结构进行了封装，使得可以直接使用 vector 容器进行向量加减法运算，从而简化了代码的编写。