# 第3章 银行家算法

## 3.1 实验目的

(1) 理解银行家算法。

(2) 掌握进程安全性检查的方法及资源分配的方法。

## 3.2 实验内容

编制模拟银行家算法的程序,并以下面给出的例子验证所编写的程序的正确性。

**【例】** 某系统有A、B、C、D四类资源,共由5个进程(P0、P1、P2、P3、P4)共享,各进程对资源的需求和分配情况如表3-1所示。

**表3-1 进程对资源的需求和分配情况**

| 进程 | 已占资源 | | | | 最大需求数 | | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | A | B | C | D |
| P0 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 2 |
| P1 | 1 | 0 | 0 | 0 | 1 | 7 | 5 | 0 |
| P2 | 1 | 3 | 5 | 4 | 2 | 3 | 5 | 6 |
| P3 | 0 | 6 | 3 | 2 | 0 | 6 | 5 | 2 |
| P4 | 0 | 0 | 1 | 4 | 0 | 6 | 5 | 6 |

现在系统中A、B、C、D四类资源分别还剩1、5、2、0个,请按银行家算法回答下列问题:

(1) 现在的系统是否处于安全状态?

(2) 如果现在进程P1提出需要(0,4,2,0)个资源的请求,系统能否满足它的请求?

## 3.3 实验准备

在该实验中涉及银行家算法和安全性检查算法,下面分别对两种算法进行具体的介绍。

**1. 银行家算法**

在避免死锁的方法中,所施加的限制条件较弱,有可能获得令人满意的系统性能。在银行家算法中把系统的状态分为安全状态和不安全状态,只要能使系统始终都处于安全

状态,便可以避免发生死锁。

银行家算法的基本思想是在分配资源之前判断系统是否是安全的,若安全才分配。它是最具有代表性的避免死锁的算法。

假设进程 Pi 提出请求 Request[i],则银行家算法按如下步骤进行判断。

Step1:如果 Request[i]<=Need[i],则转向 Step2;否则出错。

Step2:如果 Request[i]<=Available[i],则转向 Step3;否则出错。

Step3:系统试探性地分配相关资源,修改相关数据。

Available[i]=Available[i]-Request[i],

Allocation[i]=Allocation[i]+Request[i],

Need[i]=Need[i]-Request[i]。

Step4:系统执行安全性检查,如安全则分配成立;否则此前试探性分配的资源作废,系统恢复原状,进程进入等待状态。

根据以上的银行家算法步骤,可得出图 3-1 所示的程序流程图。

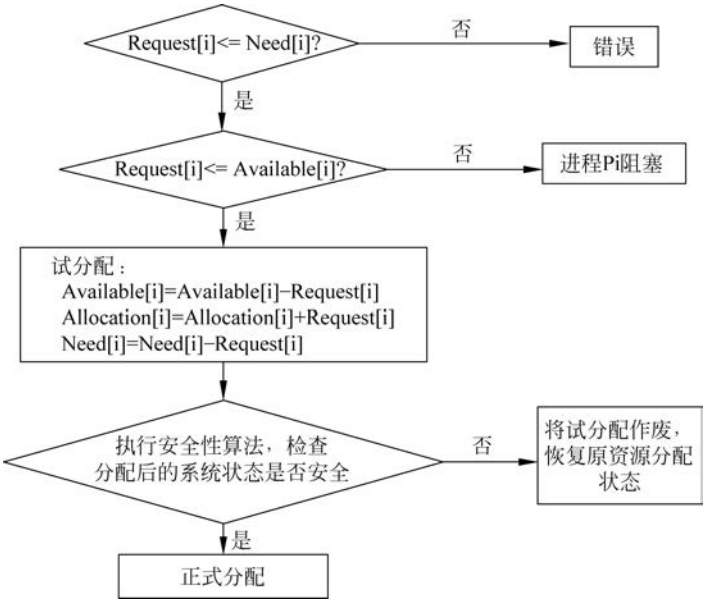


图 3-1 银行家算法程序流程图

**2. 安全性检查算法**

安全性检查算法主要是根据银行家算法进行资源分配后,检查资源分配后的系统状态是否处于安全状态之中。具体算法步骤如下所示。

Step1:设置两个工作向量——Work[i]=Available,Finish[i]=false。

Step2:从进程集合中找到一个满足下述条件的进程。

```
Finish[i] = false,
Need[i]< = Work.
```

如果能够找到该进程，则执行 Step3，否则执行 Step4。

Step3：假设上述找到的进程获得了资源，可顺利执行，直至完成，从而释放资源。

Work[i]＝Work[i]＋Allocation[i]，

Finish[i]＝true，

Goto Step2。

Step4：如果所有进程的 Finish[i]＝true，则表示该系统安全；否则表示系统不安全。

根据以上安全性检查算法的步骤，可得出图 3-2 所示的程序流程图。

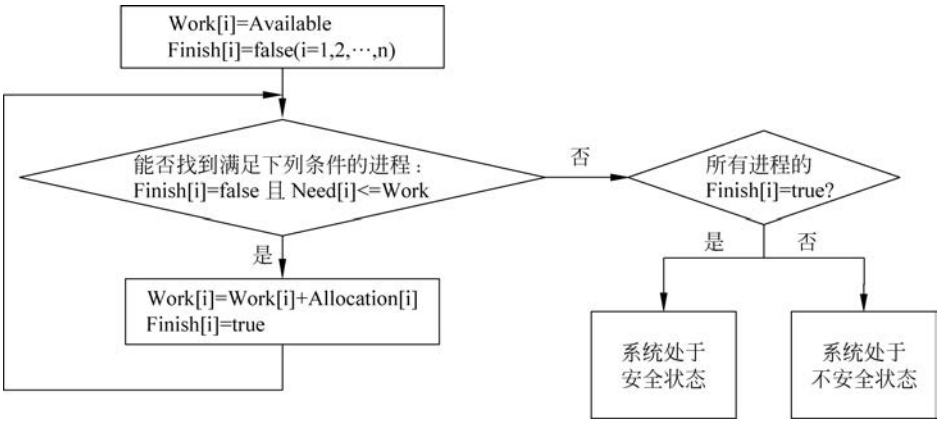


图 3-2　安全性检查算法程序流程图

## 3.4　程 序 示 例

```
#include<iostream.h>
int Available[100];                    //可利用资源数组
int Max[50][100];                      //最大需求矩阵
int Allocation[50][100];               //分配矩阵
int Need[50][100];                     //需求矩阵
int Request[50][100];
int Finish[50];
int p[50];
int m,n;                               //m 个进程,n 个资源
int IsSafe()
{

    int i,j,l=0;
    int Work[100];
    for(i=0;i<n;i++)
       Work[i]=Available[i];
    for(i=0;i<m;i++)
       Finish[i]=0;
    for(i=0;i<m;i++)
```

```
        {
           if(Finish[i] == 1) continue;
           else
           {
               for(j = 0;j < n;j++)
               {
                   if(Need[i][j]> Work[j]) break;
               }
               if(j == n)
               {
                   Finish[i] = 1;
                   for(int k = 0;k < n;k++)
                       Work[k] += Allocation[i][k];
                       p[l++] = i;
                       i = -1;
               }
               else continue;
          }
          if(l == m)
          {
              cout <<"系统是安全的 "<<'\n';
              cout <<"安全序列是 :\n";
              for(i = 0;i < l;i++)
              {
                  cout << p[i];
                  if(i!= l - 1) cout <<" -->";
              }
              cout <<'\n';
              return 1;
          }
      }
  }
  int main()                                    //银行家算法
  {
      int i,j,mi;
      cout <<" 输入进程的数目:\n";
      cin >> m;
      cout <<"输入资源的种类:\n";
      cin >> n;
      cout <<"输入每个进程最多所需的各资源数，按照 "<< m <<"x"<< n <<" 矩阵输入 \n";
      for(i = 0;i < m;i++)
      for(j = 0;j < n;j++)
      cin >> Max[i][j];
      cout <<"输入每个进程已分配的各资源数，也按照 "<< m <<"x"<< n <<" 矩阵输入 \n";
      for(i = 0;i < m;i++)
      {
          for(j = 0;j < n;j++)
          {
```

```
            cin >> Allocation[i][j];
            Need[i][j] = Max[i][j] - Allocation[i][j];
            if(Need[i][j]< 0)
            {
                cout <<"你输入的第 "<< i + 1 <<" 个进程所拥有的第 "<< j + 1 <<" 个资源数错
误，请重新输入 :\n";
                j -- ;
                continue;
            }
        }
    }
    cout <<"请输入各个资源现有的数目 :\n";
    for(i = 0;i < n;i++)
    cin >> Available[i];
    IsSafe();
    while(1)
    {
        cout <<"输入要申请资源的进程号 ( 注：第 1 个进程号为 0，以此类推 )\n";
        cin >> mi;
        cout <<"输入进程所请求的各资源的数量 \n";
        for(i = 0;i < n;i++)
           cin >> Request[mi][i];
        for(i = 0;i < n;i++)
        {
            if(Request[mi][i]> Need[mi][i])
            {
                cout <<"你输入的请求数超过进程的需求量 !\n";
                return 0;
            }
            if(Request[mi][i]> Available[i])
            {
                cout <<"你输入的请求数超过系统有的资源数 !\n";
                return 0;
            }
        }
        for(i = 0;i < n;i++)
        {
            Available[i] -= Request[mi][i];
            Allocation[mi][i] += Request[mi][i];
            Need[mi][i] -= Request[mi][i];
        }
        if(IsSafe()) cout <<"同意分配请求 !\n";
        else
        {
            cout <<"你的请求被拒绝 !\n";
            for(i = 0;i < n;i++)
            {
                Available[i] += Request[mi][i];
```

```
                Allocation[mi][i] -= Request[mi][i];
                Need[mi][i] += Request[mi][i];
            }
        }
        for(i = 0;i < m;i++)
            Finish[i] = 0;
        char YesOrNo;
        cout <<"你还想再次请求分配吗 ? 是请按 y/Y, 否按 n/N, 再确定 \n";
        while(1)
        {
            cin >> YesOrNo;
            if(YesOrNo == 'y'||YesOrNo == 'Y'||YesOrNo == 'n'||YesOrNo == 'N') break;
            else
            {
                cout <<"请按要求输入 !\n";
                continue;
            }
        }
        if(YesOrNo == 'y'||YesOrNo == 'Y') continue;
        else break;
    }
}
```

## 3.5 实 验 结 果

实验结果如图 3-3 所示。

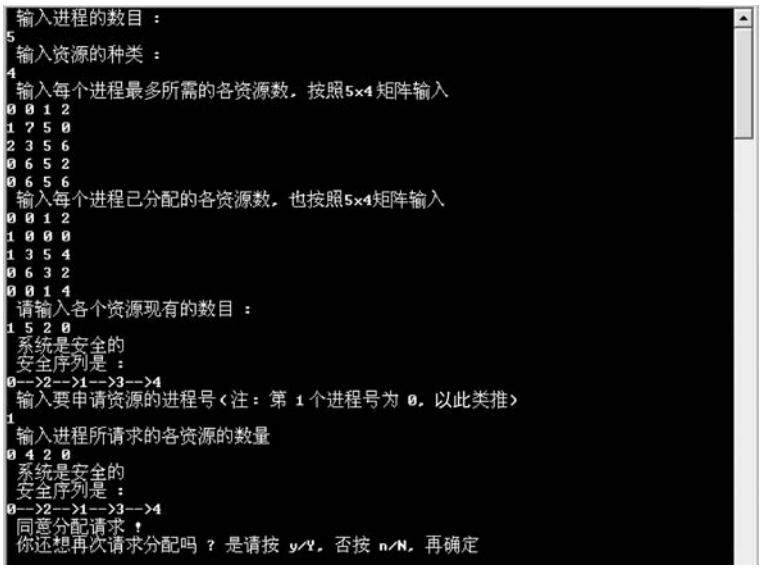


图 3-3 实 验 结 果

通过运行程序发现,例子当中的系统处于安全状态,进程 P1 提出的请求能够实现。