评阅教师	设计成绩	评阅日期

海南大学计算机与网络空间安全学院

《操作系统》课程设计报告

班　　级：

成　　员：江雪雨

指导老师：_____

完成日期：2019.12.20

死锁避免与检测

摘要：本次课程设计的内容是实现死锁的避免与检验。在操作系统中，由于多个进程对资源的争夺，对系统的安全造成威胁可能造成死锁。而死锁产生的必要条件分别为互斥条件、请求和保持条件、不可抢占条件、循环等待条件。为了解决死锁这一问题，目前处理死锁的方法为：预防死锁、避免死锁、检测死锁、解除死锁。

本次主要设计的是死锁避免和检验的算法实现。死锁避免的主要的内容是银行家算法，它包括对申请进程的预分配以及安全性检测。死锁检验的内容主要是对资源有序图的简化，当且仅当资源分配图是不可简化时，则说明系统存在死锁。

在设计银行家算法时要定义 Available(可利用的资源量)、Allocation(已分配的资源量)、Need(所需求的资源量)、Max(最大需求量)，还会使用到数组，有利于各个变量矩阵的运算。

死锁避免算法的结果可以动态低显示系统为每个进程分配资源时各个资源矩阵的变化，结果显示为是否处于安全状态，若处于安全状态，则出现安全序列，否则显示系统处于不安全状态。死锁检测算法动态地显示有向边的资源分配图的简化过程，结果显示当前环境不会发生死锁。

关键词：银行家算法； 资源分配图；避免死锁； 操作系统；检测死锁

1 设计任务

1.1 设计目的

本次课程设计主要目的是掌握银行家算法，以及加深对避免死锁的方法和死锁检测的理解，推近对死锁更深层次的掌握。

1.2 设计内容

1.2.1 设计一个程序演示死锁避免算法（银行家算法），该演示程序能显示各进程申请和

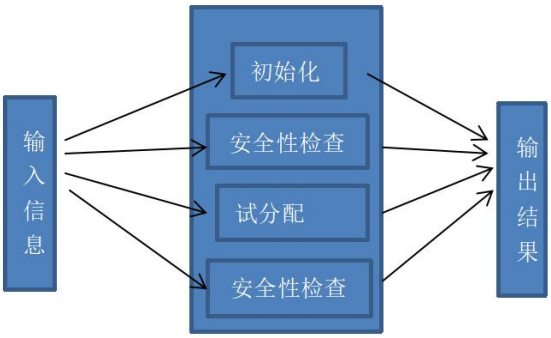
释放资源以及系统动态分配资源的过程，便于用户观察和分析；如果系统不能分配，也应给出系统进入不安全状态”的提示，已随机给出进程和各类共享资源数量、已分配量、申请量。

1.2.2 设计一个程序演示教材 P116-117 给出的死锁检验算法，该演示程序能显示资源分配图的简化过程，并给出最终检测结果（系统死锁还是不死锁），已随机给出进程和各类共享资源数量、已分配量、申请量。

2 总体设计

2.1 总体结构

2.1.1 死锁避免算法



2.1.2 死锁检测算法

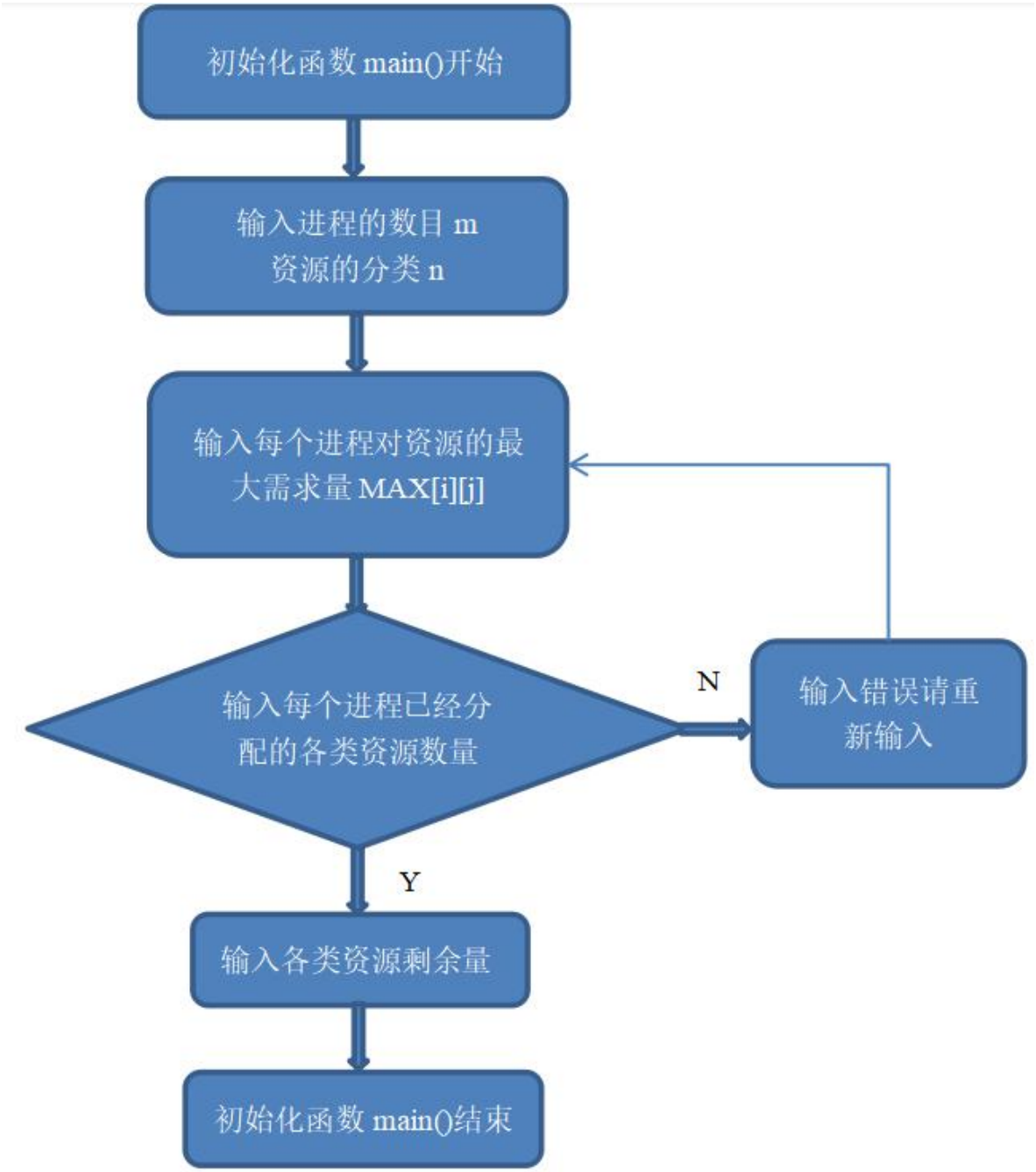
2.2 开发环境

Dev-c++ 5.11 开发工具、Windows 10 系统

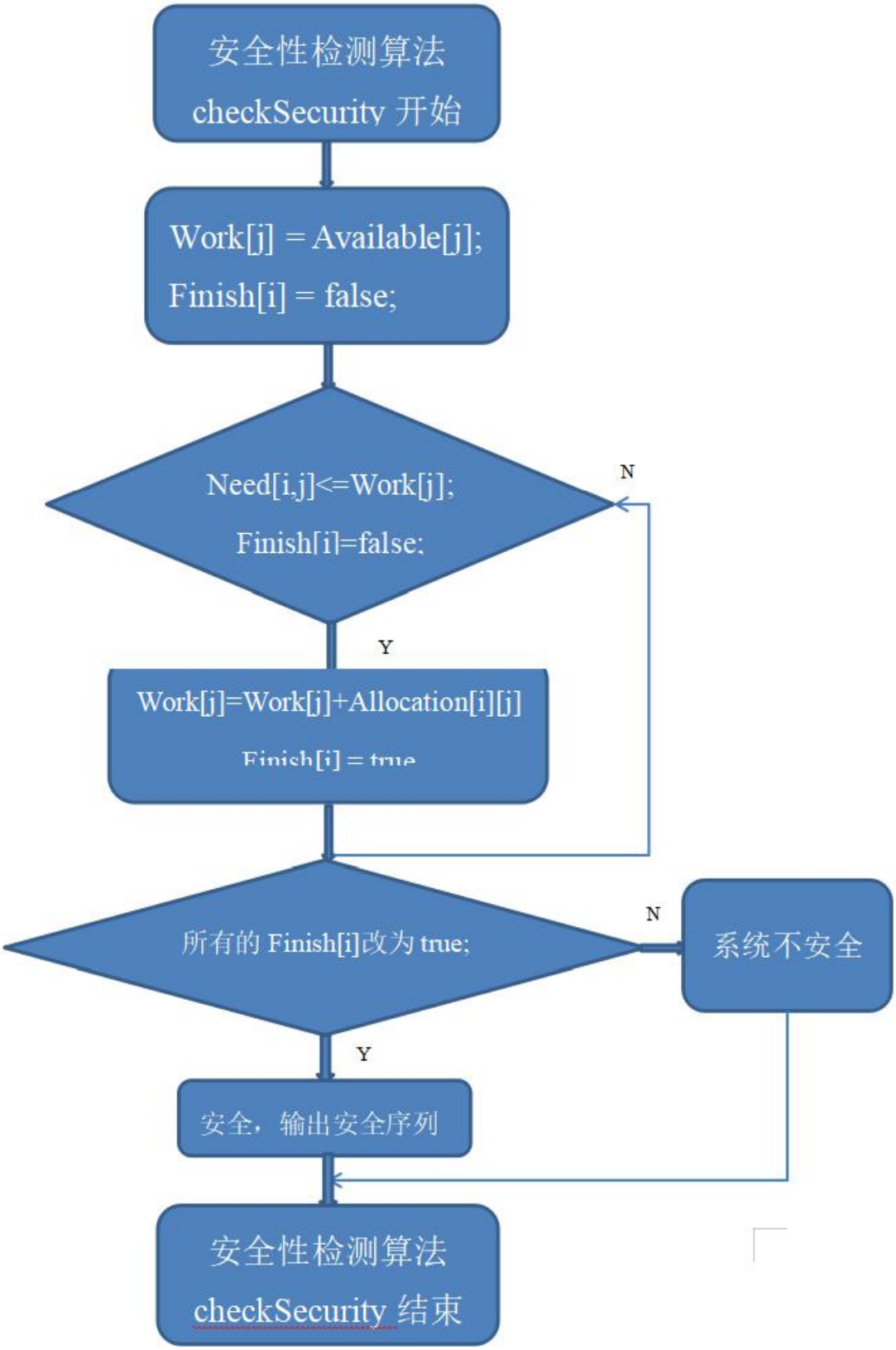
3 算法设计

3.1 死锁避免算法设计

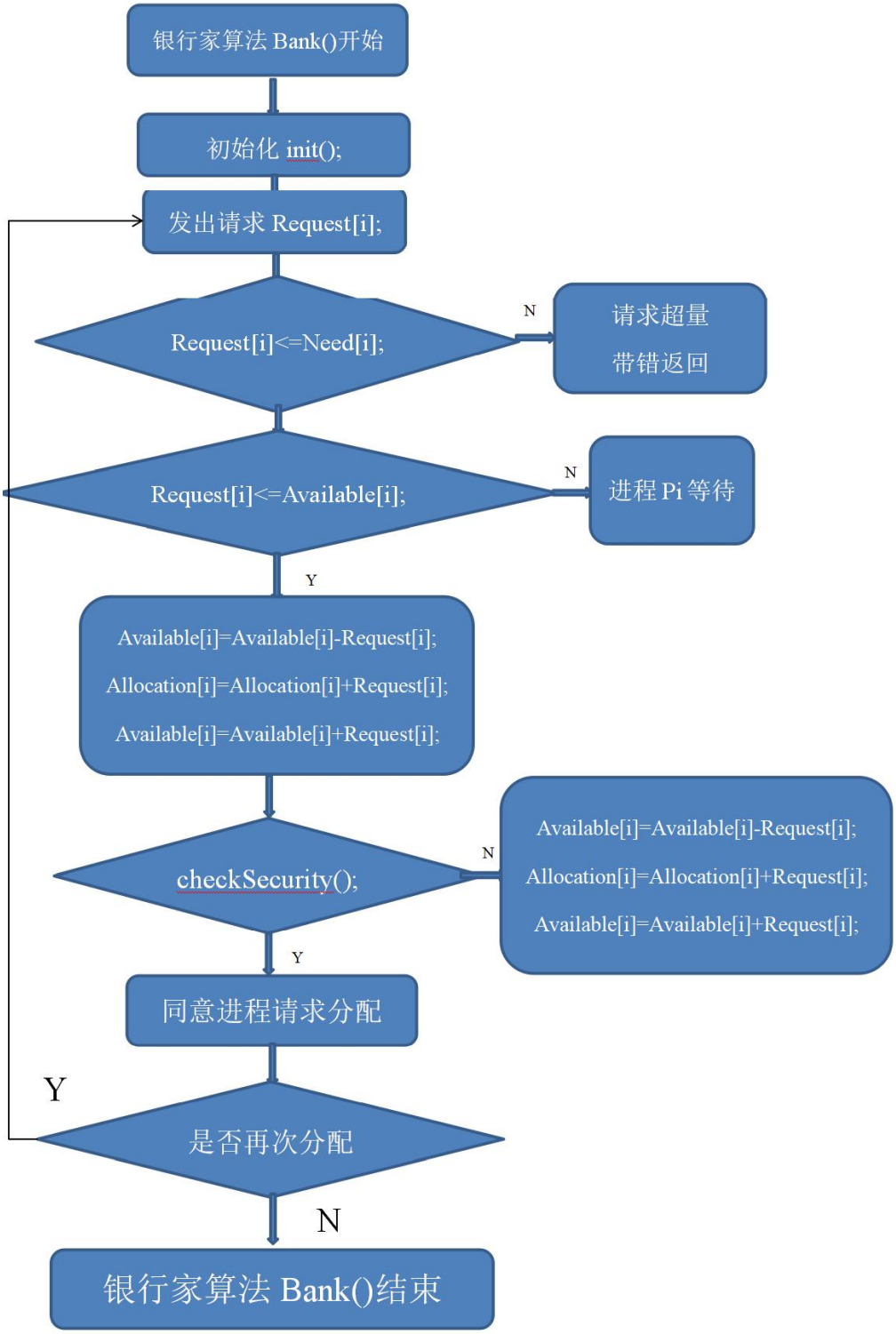
3.1.1 初始化算法流程图



3.1.2 安全性检测算法流程图



3.1.3 银行家算法流程图



银行家算法要先给定进程数和资源数，在随机输入符合要求的 Allocation，Max，Need 矩阵，给出 Available 向量，完成初始化。当有进程申请资源时要与 Need 和可用的 Available 相对，如果满足则进行预分配，再利用安全性算法进行检查。如果系统处于安全状态，则为进程分配资源，否则拒绝为该进程分配资源。

3.2 动态资源分配

定义 Allocation，Max，Need 矩阵，给出 Available，完成初始化，对每个进程进行 Need 与 Available 进行比较，循环显示资源动态分配图简化的过程，即进程删边的过程，资源分配图的简化是把进程有向边删完，最后证明该环境是安全的，意味着该资源分配图无环路。

4 系统实现

4.1 死锁避免算法

4.1.1 初始化模块实现

```
//初始化算法
void Init()
{
    int i,j;
    printf ("请输入进程的数目:\n");
    scanf ("%d",&m);
    printf ("请输入资源的种类:\n");
    scanf ("%d",&n);
    printf ("请输入每个进程对资源的最大需求量,按照%d*%d矩阵输入:\n",m,n);
     for (i=0;i<m;++i)
      {
        for (j=0;j<n;++j)
         scanf ("%d",&Max[i][j]);
      }
    printf ("请输入每个进程已经分配的各类资源数量,按照%d*%d矩阵输
入:\n",m,n);
    for (i=0;i<m;++i)
      {
        for (j=0;j<n;++j)
          {
            scanf ("%d",&Allocation[i][j]);

          }
         }
     printf ("获得Need矩阵的值为:\n");
     {
         for (i=0;i<m;++i)
         {
            for (j=0;j<n;++j)
            {
                Need[i][j] = Max[i][j]-Allocation[i][j];
                printf ("%d ",Need[i][j]);
            }
            printf ("\n");

            if (Need[i][j]<0)
           {
              printf ("第%d行第%d个资源错误，请重新输入:\n",i+1,j+1);
             j--;
             continue;
          }
         }

      }

    printf ("请输入各类资源剩余量:\n");
     for (i=0;i<n;++i)
      scanf ("%d",&Available[i]);

    printf ("进程的全部信息如下：\n");
    printf ("进程名\tMax\t\tAllocation\tNeed\n");
    for (int i=0;i<m;++i)
    {
        printf ("P%d\t",i);
        for (int j=0;j<n;++j)
        {
            printf ("%d ",Max[i][j]);
        }
        printf ("\t");
            for (int j=0;j<n;++j)
        {
            printf ("%d ",Allocation[i][j]);
        }
        printf ("\t");
            for (int j=0;j<n;++j)
        {
            printf ("%d ",Need[i][j]);
        }
        printf ("\n");
    }
```

4.1.2 安全性检查模块实现

```
//安全性检测算法
bool checkSecurity (int Flag)
{
    int i,j,k,l=0;
    int Work[100];//可以用的资源数组；
    for (i=0;i<n;++i)
    {
        Work[i] = Available[i];
    }
    for (i=0;i<m;++i)
    {
        Finish[i]=false;//Finish 记录每个进程是否安全
    }

    for (i=0;i<m;++i)
    {
        if (Finish[i]==true)
          continue;
        else
         {
            for (j=0;j<n;++j)//循环查找第i个进程所需的各个资源数是否超过系统现有的第j个资源数
            {
                if (Need[i][j]>Work[j])//如果第i个进程所需的第j个资源数超过系统现有的第j个资源数，则拒绝申请；
                    break;
            }

            if (j == n)//如果第i个进程所需的各个资源数没有超过系统现有的资源数；
            {
                Finish[i]=true;
                for (k=0;k<n;++k)
                    Work[k]=Work[k]+Allocation[i][k];//将第i个进程各个已分配资源数+系统有的对应资源数赋值给Work；
                p[l++]=i;
                 i=-1;// 记录进程号；
            }
            else //如果超过则继续循环执行下一个过程
            {
                continue;
            }
        }

        if (Flag==0)
        {
            if (l == m)//如果所有的进程都能够被满足运行时；
            {
                                            //show();
                printf ("系统是安全的！\n");
                printf ("安全序列为:\n");
                for (i=0;i<l;++i)//显示资源分配给进程的顺序；
                {
                    printf ("%d",p[i]);
                    if(i != l-1)//输出箭头
                    printf ("-->");
                }
                printf ("\n");
                return true;
            }
            else
            {
                printf ("系统是不安全的!\n");
                return false;
            }

        }
        else if (Flag==1)
        {
            show ();
        }


    }//for循环
        return 0;
}
```

4.1.3 银行家算法模块的实现

```
//银行家算法的实现
int Bank(int Flag)
{

    while (1)
        {
            int mi,i;
            printf ("请输入要申请资源的进程号:  (第一个进程号为0,第二个进程号为1,以此类推)\n");
            scanf ("%d",&mi);
            printf ("请输入进程所请求得各个资源的数量:\n");
            for (i=0;i<n;++i)
             {
                scanf ("%d",&Request[mi][i]);
             }
            for (i=0;i<n;++i)
             {
                if (Request[mi][i]>Need[mi][i])
                 {
                    printf ("所请求的资源超过进程进程的需求量! \n");
                    return 0;
                 }
                if (Request[mi][i]>Available[i])
                 {
                    printf ("所请求的资源超过系统现有的资源数! \n");
                    return 0;
                 }

             }
            for (i=0;i<n;++i)
             {
                if ( Request[mi][i] <= Need[mi][i] && Request[mi][i] <= Available[i] )
                 {
                    Available[i] = Available[i]-Request[mi][i];
                    Allocation[mi][i] = Allocation[mi][i]+Request[mi][i];
                    Need[mi][i] = Need[mi][i]-Request[mi][i];
                 }
                if (checkSecurity (Flag))
                 {
                    printf (">_<同意您的分配请求! \n");
                 }
                else
                 {
                    printf ("T_T您的请求被拒绝! \n");
                    for (i=0;i<n;++i)
                     {
                       Available[i] = Available[i]-Request[mi][i];
                       Allocation[mi][i] = Allocation[mi][i]+Request[mi][i];
                       Need[mi][i] = Need[mi][i]-Request[mi][i];
                     }
                 }
                for (i=0;i<m;i++)
                 {
                     Finish[i]=false;
                 }

                char Fl;//设置一个标志位 ;
                printf ("是否再次请求分配? 是请按Y/有, 否请按N/n\n");
                while(1)
                 {
                    scanf ("%c\n",&Fl);
                    if (Fl == 'Y'||Fl == 'y'||Fl == 'N'||Fl == 'n')
                     break;
                    else
                     {
                        printf ("请重新输入:\n");
                        continue;
                     }
                 }
                 if (Fl == 'Y'||Fl == 'y')
                   continue;
                else
                   break;
         }
    }
}
```

4.2 动态资源分配

4.2.1 初始化算法

```
//初始化算法
void Init()
{
    int i,j;
    printf ("请输入进程的数目:\n");
    scanf ("%d",&m);
    printf ("请输入资源的种类:\n");
    scanf ("%d",&n);
    printf ("请输入每个进程对资源的最大需求量,按照%d*%d矩阵输入:\n",m,n);
     for (i=0;i<m;++i)
      {
        for (j=0;j<n;++j)
         scanf ("%d",&Max[i][j]);
      }
    printf ("请输入每个进程已经分配的各类资源数量,按照%d*%d矩阵输
入:\n",m,n);
      for (i=0;i<m;++i)
      {
        for (j=0;j<n;++j)
          {
            scanf ("%d",&Allocation[i][j]);

          }
        }
     printf ("获得Need矩阵的值为:\n");
     {
         for (i=0;i<m;++i)
         {
            for (j=0;j<n;++j)
            {
                Need[i][j] = Max[i][j]-Allocation[i][j];
                printf ("%d ",Need[i][j]);
            }
            printf ("\n");

            if (Need[i][j]<0)
           {
              printf ("第%d行第%d个资源错误，请重新输入:\n",i+1,j+1);
             j--;
             continue;
          }
         }

      }

    printf ("请输入各类资源剩余量:\n");
     for (i=0;i<n;++i)
      scanf ("%d",&Available[i]);

    printf ("进程的全部信息如下：\n");
    printf ("进程名\tMax\t\tAllocation\tNeed\n");
    for (int i=0;i<m;++i)
    {
        printf ("P%d\t",i);
        for (int j=0;j<n;++j)
        {
            printf ("%d ",Max[i][j]);
        }
        printf ("\t");
            for (int j=0;j<n;++j)
        {
            printf ("%d ",Allocation[i][j]);
        }
        printf ("\t");
            for (int j=0;j<n;++j)
        {
            printf ("%d ",Need[i][j]);
        }
        printf ("\n");
    }
```

4.2.2 检验安全算法

```
//安全性检测算法
bool checkSecurity (int Flag)
{
    int i,j,k,l=0;
    int Work[100];//可以用的资源数组；
    for (i=0;i<n;++i)
    {
        Work[i] = Available[i];
    }
    for (i=0;i<m;++i)
    {
        Finish[i]=false;//Finish 记录每个进程是否安全
    }

    for (i=0;i<m;++i)
    {
        if (Finish[i]==true)
          continue;
        else
         {
            for (j=0;j<n;++j)//循环查找第i个进程所需的各个资源数是否超过系统现有的第j个资源数
            {
                if (Need[i][j]>Work[j])//如果第i个进程所需的第j个资源数超过系统现有的第j个资源数，则拒绝申请；
                    break;
            }

            if (j == n)//如果第i个进程所需的各个资源数没有超过系统现有的资源数；
            {
                Finish[i]=true;
                for (k=0;k<n;++k)
                    Work[k]=Work[k]+Allocation[i][k];//将第i个进程各个已分配资源数+系统有的对应资源数赋值给Work；
                p[l++]=i;
                 i=-1;// 记录进程号；
            }
            else //如果超过则继续循环执行下一个过程
            {
                continue;
            }
        }

        if (Flag==0)
        {
            if (l == m)//如果所有的进程都能够被满足运行时；
            {
                                            //show();
                printf ("系统是安全的！\n");
                printf ("安全序列为:\n");
                for (i=0;i<l;++i)//显示资源分配给进程的顺序；
                {
                    printf ("%d",p[i]);
                    if(i != l-1)//输出箭头
                    printf ("-->");
                }
                printf ("\n");
                return true;
            }
            else
            {
                printf ("系统是不安全的!\n");
                return false;
            }

        }
        else if (Flag==1)
        {
            show ();
        }


    }//for循环
        return 0;
}
```

4.2.3 动态显示资源简化算法

```
void show()
{
    int i;
    for(i=0;i<m;i++) //当前系统可满足%d进程的需求，分配资源给该进程。进程运行结束后，系统收回该进程%d的资源。
    printf("第%d步，删除进程%d的边\n",i+1,p[i]);
    printf ("当前环境不会发生死锁！\n");
}
```

5 系统测试

5.1 银行家算法完整代码段和死锁检测完整代码

```
#include <stdio.h>

#define M 100                    // //全局变量定义

#define N 50                     //定义 M 个进程，N 类资源

void Init();

bool checkSecurity (int);

void show();

int Bank(int);

int Available[M];              //可利用资源数组

int Max[N][M];      //最大需求矩阵

int Allocation[N][M];    //分配矩阵

int Need[N][M];                //需求矩阵

int Request[N][M];          //M 个进程还需要 N 类资源的资源量

bool Finish[N];

int p[N];

int m,n;      //M 个进程,N 类资源

int main (void)
```

```
{
        int i;
        int Flag0 = 0;
        int Flag1 = 1;
        Init ();
        while(1)
        {

        printf(" 1 -->  银行家算法。\n");
        printf(" 2 -->  死锁检测。\n") ;
        printf(" 3 -->  退出本次实验。\n");
        printf("     请选择你要进行的操作：");

        scanf("%d",&i);

        switch(i)
        {
        case 1:
          Bank(Flag0);
          //checkSecurity (Flag0);
          break;
```

```
            case 2:
              checkSecurity (Flag1);
              break;

            case 3:
              return 0;

            }
            }
}
//初始化算法
void Init()
{
            int i,j;
            printf ("请输入进程的数目:\n");
            scanf ("%d",&m);
            printf ("请输入资源的种类:\n");
            scanf ("%d",&n);
            printf ("请输入每个进程对资源的最大需求量,按照%d*%d 矩阵输入:\n",m,n);
              for (i=0;i<m;++i)
                {
                    for (j=0;j<n;++j)
```

```
            scanf ("%d",&Max[i][j]);
        }
    printf ("请输入每个进程已经分配的各类资源数量,按照%d*%d 矩阵输入:\n",m,n);
        for (i=0;i<m;++i)
        {
            for (j=0;j<n;++j)
                {
             scanf ("%d",&Allocation[i][j]);


                }
            }
        printf ("获得 Need 矩阵的值为:\n");
        {
          for (i=0;i<m;++i)
          {
            for (j=0;j<n;++j)
            {
                Need[i][j] = Max[i][j]-Allocation[i][j];
                 printf ("%d ",Need[i][j]);
            }
            printf ("\n");
```

```
            if (Need[i][j]<0)
          {
          printf ("第%d 行第%d 个资源错误，请重新输入:\n",i+1,j+1);
            j--;
            continue;
      }
        }

   }

printf ("请输入各类资源剩余量:\n");
  for (i=0;i<n;++i)
    scanf ("%d",&Available[i]);

printf ("进程的全部信息如下：\n");
printf ("进程名\tMax\t\tAllocation\tNeed\n");
for (int i=0;i<m;++i)
{
   printf ("P%d\t",i);
   for (int j=0;j<n;++j)
   {
        printf ("%d ",Max[i][j]);
```

```
            }
            printf ("\t");
                for (int j=0;j<n;++j)
            {
                printf ("%d ",Allocation[i][j]);
            }
            printf ("\t");
                for (int j=0;j<n;++j)
            {
                printf ("%d ",Need[i][j]);
            }
            printf ("\n");
        }
        printf ("目前可利用的资源量 Available:\n");
        for (int i=0;i<n;i++)
        {
            printf ("%d",Available[i]);
        }
        printf ("\n");
}
//安全性检测算法
bool checkSecurity (int Flag)
```

```
{
        int i,j,k,l=0;
        int Work[100];//可以用的资源数组；
        for (i=0;i<n;++i)
        {
          Work[i] = Available[i];
        }
        for (i=0;i<m;++i)
        {
          Finish[i]=false;//Finish 记录每个进程是否安全
    }

        for (i=0;i<m;++i)
        {
          if (Finish[i]==true)
             continue;
          else
            {
                for (j=0;j<n;++j)//循环查找第 i 个进程所需的各个资源数是否超过系统现有的第 j 个资源数
                {
                    if (Need[i][j]>Work[j])//如果第 i 个进程所需的第 j 个资源数超过系统现有的第 j 个资源数，则拒绝申请；
```

```
                    break;
                }

            if (j == n)//如果第 i 个进程所需的各个资源数没有超过系统现有的资源数；
            {
                Finish[i]=true;
                for (k=0;k<n;++k)
                        Work[k]=Work[k]+Allocation[i][k];//将第 i 个进程各个已分配资源数+系统有的对应资源数赋值给 Work;
                p[l++]=i;
                 i=-1;// 记录进程号；
                }
            else //如果超过则继续循环执行下一个过程
            {
                continue;
            }
        }

    if (Flag==0)
        {
            if (l == m)//如果所有的进程都能够被满足运行时；
             {
```

```
                                        //show();
            printf ("系统是安全的！ \n");
            printf ("安全序列为:\n");
            for (i=0;i<l;++i)//显示资源分配给进程的顺序；
            {
                printf ("%d",p[i]);
                if(i != l-1)//输出箭头
                printf ("-->");
            }
            printf ("\n");
            return true;
        }
        else
        {
             printf ("系统是不安全的!\n");
                return false;
        }

}
else if (Flag==1)
{
    show ();
```

```
            }


    }//for 循环
                return 0;
}
//银行家算法的实现
int Bank(int Flag)
{

    while (1)
                {
                 int mi,i;
                 printf ("请输入要申请资源的进程号:  (第一个进程号为 0,第二个进程号为 1,以此类推)\n");
                 scanf ("%d",&mi);
                 printf ("请输入进程所请求得各个资源的数量:\n");
                 for (i=0;i<n;++i)
                   {
                      scanf ("%d",&Request[mi][i]);
                   }
                 for (i=0;i<n;++i)
                   {
```

```
        if (Request[mi][i]>Need[mi][i])
          {
            printf ("所请求的资源超过进程进程的需求量！\n");
            return 0;
          }
        if (Request[mi][i]>Available[i])
          {
            printf ("所请求的资源超过系统现有的资源数！\n");
            return 0;
          }

      }
    for (i=0;i<n;++i)
      {
        if ( Request[mi][i] <= Need[mi][i] && Request[mi][i] <=
Available[i] )
          {
            Available[i] = Available[i]-Request[mi][i];
            Allocation[mi][i] = Allocation[mi][i]+Request[mi][i];
            Need[mi][i] = Need[mi][i]-Request[mi][i];
          }
        if (checkSecurity (Flag))
          {
```

```
    printf (">_<同意您的分配请求！\n");
  }
else
  {
    printf ("T_T 您的请求被拒绝！\n");
    for (i=0;i<n;++i)
      {
        Available[i] = Available[i]-Request[mi][i];
        Allocation[mi][i] = Allocation[mi][i]+Request[mi][i];
        Need[mi][i] = Need[mi][i]-Request[mi][i];
      }
  }
for (i=0;i<m;i++)
  {
     Finish[i]=false;
  }

char Fl;//设置一个标志位 ；
printf ("是否再次请求分配？是请按 Y/有，否请按 N/n\n");
while(1)
  {
    scanf ("%c\n",&Fl);
```

```
                    if (Fl == 'Y'||Fl == 'y'||Fl == 'N'||Fl == 'n')
                      break;
                    else
                      {
                        printf ("请重新输入:\n");
                        continue;
                      }
                  }
                  if (Fl == 'Y'||Fl == 'y')
                    continue;
                  else
                    break;
              }
      }
}

void show()
{
        int i;
        for(i=0;i<m;i++) //当前系统可满足%d 进程的需求，分配资源给该进程。进程运行结束后，系统收回该进程%d 的资源。
          printf("第%d 步，删除进程%d 的边\n",i+1,p[i]);
```

```
            printf ("当前环境不会发生死锁！\n");
}
```

5.2 初始化模块运行结果

```
请输入进程的数目:
5
请输入资源的种类:
4
请输入每个进程对资源的最大需求量,按照5*4矩阵输入:
0 0 1 2
1 7 5 0
2 3 5 6
0 6 5 2
0 6 5 6
请输入每个进程已经分配的各类资源数量,按照5*4矩阵输入:
0 0 1 2
1 0 0 0
1 3 5 4
0 6 3 2
0 0 1 4
获得Need矩阵的值为:
0 0 0 0
0 7 5 0
1 0 0 2
0 0 2 0
0 6 4 2
请输入各类资源剩余量:
1 5 2 0
进程的全部信息如下:
进程名  Max             Allocation      Need
P0      0 0 1 2         0 0 1 2         0 0 0 0
P1      1 7 5 0         1 0 0 0         0 7 5 0
P2      2 3 5 6         1 3 5 4         1 0 0 2
P3      0 6 5 2         0 6 3 2         0 0 2 0
P4      0 6 5 6         0 0 1 4         0 6 4 2
目前可利用的资源量Available:
1520
```

5.3 银行家算法测试结果

```
 1 --> 银行家算法。
 2 --> 死锁检测。
 3 --> 退出本次实验。
   请选择你要进行的操作：1
请输入要申请资源的进程号：（第一个进程号为0,第二个进程号为1,以此类推）
1
请输入进程所请求得各个资源的数量:
0 0 1 0
系统是不安全的!
T_T您的请求被拒绝!
是否再次请求分配？是请按Y/有，否请按N/n
n
请重新输入:
2
请输入要申请资源的进程号：（第一个进程号为0,第二个进程号为1,以此类推）
请输入进程所请求得各个资源的数量:
3
1 1 1
所请求的资源超过进程进程的需求量!
```

5.4 动态显示资源简化图算法结果

```
 1 --> 银行家算法。
 2 --> 死锁检测。
 3 --> 退出本次实验。
   请选择你要进行的操作：2
第1步，删除进程0的边
第2步，删除进程0的边
第3步，删除进程0的边
第4步，删除进程0的边
第5步，删除进程0的边
当前环境不会发生死锁!
第1步，删除进程0的边
第2步，删除进程2的边
第3步，删除进程0的边
第4步，删除进程0的边
第5步，删除进程0的边
当前环境不会发生死锁!
第1步，删除进程0的边
第2步，删除进程2的边
第3步，删除进程1的边
第4步，删除进程0的边
第5步，删除进程0的边
当前环境不会发生死锁!
第1步，删除进程0的边
第2步，删除进程2的边
第3步，删除进程1的边
第4步，删除进程3的边
第5步，删除进程0的边
当前环境不会发生死锁!
第1步，删除进程0的边
第2步，删除进程2的边
第3步，删除进程1的边
第4步，删除进程3的边
第5步，删除进程4的边
当前环境不会发生死锁!
```

6 设计小结

在本次课程设计中，由于没有学过数据结构等课程，一开始对于整个设计的框架比较模糊，再把课本的例题思路慢慢整理，大致建立一个框架，再往这个框架里面填入一些函数，搭建成一个个可以实现某功能的结构。比如初始化函数模块、安全性检测函数模块、银行家算法模块，另外，可以实现对时间复杂度的优化。但在实现银行家算法的过程中，由于一些函数调用以及变量命名的重复，思路卡了几次，我去请教了一些老师及同学，他们给了我一些解决方法和思路，使我把这几个问题解决了。在实现资源简化图设计的过程中，由于目前对链表等数据结构的不熟悉，我使用了一些文字循环显示资源动态分配图简化的过程，即进程删边的过程，资源分配图的简化是把进程有向边删完，最后证明该环境是安全的，意味着该资源分配图无环路。

经过这次的课程设计实验，我学到了许多的知识及熟练地掌握了银行家算法的思想，同时进一步提高了我的算法思维能力，也证明了我还有很多地知识还要不断地学习 ，不断地锻炼我的算法思维，在今后的课程学习中，我将继续前进。

参考文献

[1] 汤小丹、梁红兵等编著.《计算机操作系统》西安电子科技大学出版社 2014 年

《操作系统》课程设计成绩自行评定表

姓名	学号	职务	总成绩
江雪雨	20186807310237	个人	A