2018

# 操作系统实验报告

姓名：陶略

学号：031510319

班号：1615102

# 目录



# 写者优先

读者写者问题：

读者一写者问题是一个用信号量实现的经典进程同步问题。在系统中，一个数据集（如文件或记录）被几个并发进程共享，这些线程分两类，一部分只要求进行复操作，称之为“读者”；另一类要求写或修改操作，我们称之为“写者”。一般而言，对一个数据集，为了保证数据的完整性、正确性，允许多个读者进程同时访问，但是不允许一个写者进程同其它任何一个进程（读者或者写者）同时访问，而这类问题就称之为“读者-写者”问题。

写者优先：

读者优先的算法在操作系统相关的书籍中都有介绍，这是一种最简单的解决办法：当没有写进程正在访问共享数据集时，读进程可以进入访问，否则必须等待。而读者优先的算法存在“饿死写者”线程的问题：只要有读者不断到来，写者就要持久地等待，直到 所有的读者都读完且没有新的读者到来时写者才能写数据集。而在很多情况下我们需要避免“饿死写者”，故而采用写者优先算法。1.要让读者与写者之间、以及写者与写者之问要互斥地访同数据集；2.在无写进程到来时各读者可同时访问数据集；3.在读者和写者都等待时访问时写者优先。

算法思路：

用两个不同的互斥信号量分别实现读者与写者间的互斥及各写者进程间的互斥：以互斥信号量 Wmutex 实现各写者问的互斥，互斥信号量 Rmutex 实现各读者与写者问的互斥；设置两个整型变量 Wcount 和 Rcount 分别记录等待的写者数和正在读的读者数，因 Wcount、Rcount 都是共享变量，因此还要设置两个互斥信号量 Mutl 和 Mut2 以实现进程对这两个变最的互斥访问。

算法流程：

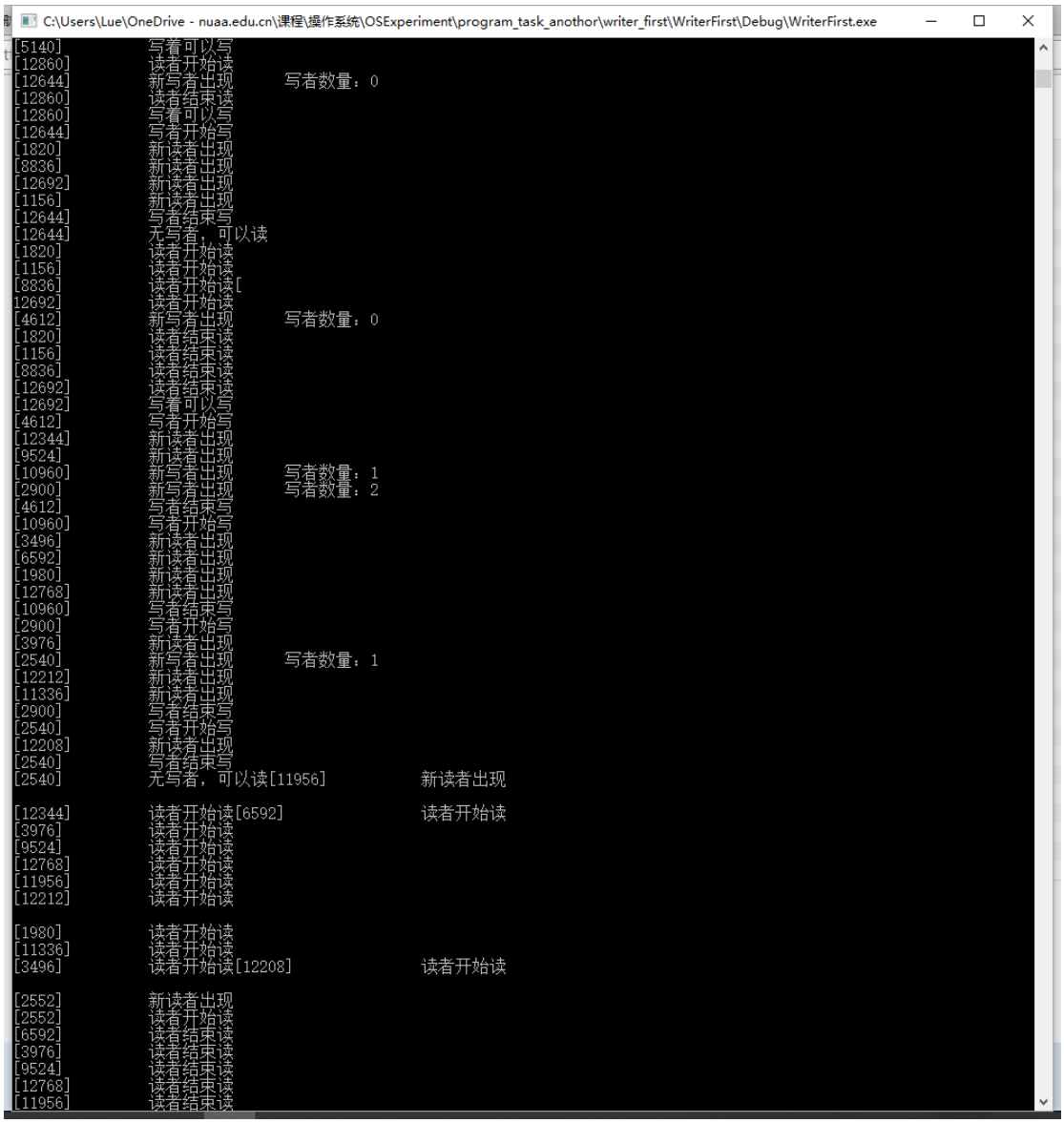
1：读者、写者均按照 Mut1、Fmutex 的顺序申请信号量。

2：写者获得 Fmutex 后，直至最后一个写者释放 Fmutex，读者均被阻塞。读者获得 Fmutex 后，直至最后一个读者(不包含由于 Mut1 被阻塞的读者)释放 Fmutex，写者均被阻塞。

3：读者申请到 Mut1 后立即释放；而写者申请到 Mut1 后，就一直占用不放，直至申请到 Fmutex。

4：Fmutex 的作用就是写者、读者间的互斥；Mut1 的额外作用就是帮助写者优先竞争到 Fmutex。

实验截图：




代码:

```
#include <Windows.h>
#include <process.h>
#include <iostream>
#include <time.h>
using namespace std;

#define P(S) WaitForSingleObject(S, INFINITE)
#define V(S) ReleaseSemaphore(S, 1, NULL)
// Fmutex   读者写者互斥
// Wmutex   写者互斥
// Mut1     Rcount 互斥 && 竞争 Fmutex
// Mut2     Wcount 互斥
HANDLE Mut1, Mut2, Wmutex, Fmutex;
```

```
int Rcount = 0;
int Wcount = 0;

DWORD WINAPI writer()
{
    DWORD id = GetCurrentThreadId();

    cout << "[" << id << "]\t\t 新写者出现" << "\t 写者数量:
" << Wcount << endl;
    P(Mut1);
    Wcount += 1;
    if (Wcount == 1)
        P(Fmutex);      // 如果有读者，写者阻塞在此处
    V(Mut1);

    P(Wmutex);
    cout << "[" << id << "]\t\t 写者开始写" << endl;
    Sleep(2000);
    cout << "[" << id << "]\t\t 写者结束写" << endl;
    V(Wmutex);

    P(Mut1);
    Wcount -= 1;
    if (Wcount == 0)
    {
        cout << "[" << id << "]\t\t 无写者，可以读" << endl;
        V(Fmutex);
    }
    V(Mut1);

    return 1;
}

DWORD WINAPI reader()
{
    DWORD id = GetCurrentThreadId();

    cout << "[" << id << "]\t\t 新读者出现" << endl;

    P(Mut1);    // 读者要先申请 Mut1，如果有写者在等待 Fmutex，则读者被阻塞，写者
优先
    V(Mut1);    // 立即释放 Mut1，使写者可以随时申请到 Mut1

    P(Mut2);
```

```
    Rcount += 1;
    if (Rcount == 1)
        P(Fmutex);  // 第一个读者进入时，申请 Fmutex：如果有写者，则第一个读者会阻塞在此处
    V(Mut2);

    cout << "[" << id << "]\t\t 读者开始读" << endl;
    Sleep(500);
    cout << "[" << id << "]\t\t 读者结束读" << endl;

    P(Mut2);
    Rcount -= 1;
    if (Rcount == 0)
    {
        cout << "[" << id << "]\t\t 写着可以写" << endl;
        V(Fmutex);      // 最后一个读者退出时，释放 Fmutex
    }
    V(Mut2);

    return 1;
}

int main()
{
    Wmutex = CreateSemaphore(NULL, 1, 1, NULL);
    Fmutex = CreateSemaphore(NULL, 1, 1, NULL);
    Mut1 = CreateSemaphore(NULL, 1, 1, NULL);
    Mut2 = CreateSemaphore(NULL, 1, 1, NULL);

    // srand((unsigned)time(NULL));
    srand(90);

    while (true)
    {
        Sleep(500);

        int rC = rand() % 1000;
        if (rC % 6 == 0)
            CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)writer, NULL, NULL, NULL);
        else
            CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)reader, NULL, NULL, NULL);
    }
```

```
97.
98.     return 0;
99. }
```

## 多线程画圆、画方

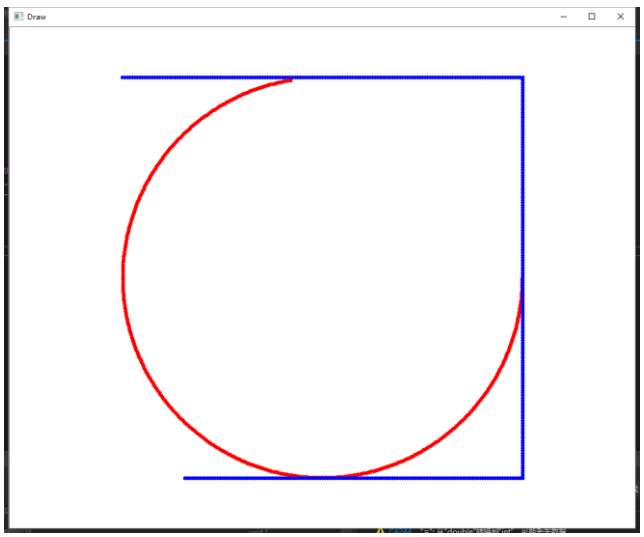
题目要求：

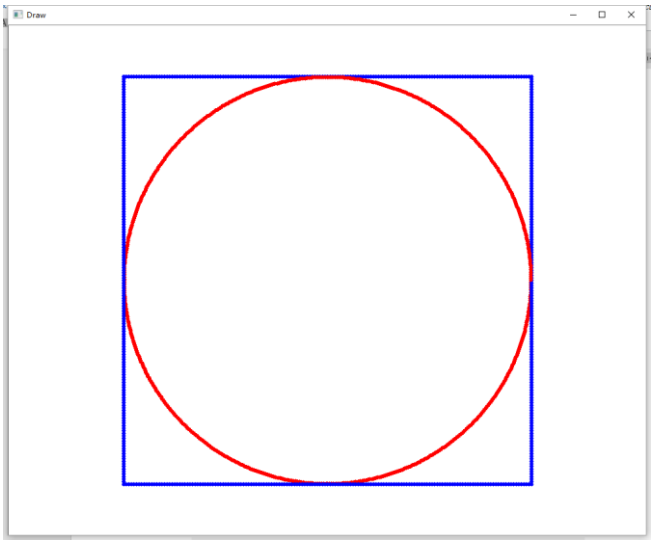
在 Windows 系统下创建两个线程，一个线程用来画圆，另一个画方，主线程负责实时显示图形。

实现思路：

利用 opencv 的 Mat 类来储存图形，利用一个信号量 mutex 来保证对图形的互斥访问，两个线程分别画圆和画方，画之前申请 mutex，画之后释放 mutex。

实验结果：




代码:

```
// drawCircle.cpp: 定义控制台应用程序的入口点。
//

#include "stdafx.h"


#include <opencv2/opencv.hpp>
#include <iostream>
#include <Windows.h>
#include <process.h>

using namespace std;
using namespace cv;

#define PI 3.14159265358979
#define P(S) WaitForSingleObject(S, INFINITE)
#define V(S) ReleaseSemaphore(S, 1, NULL)
HANDLE mutex;
Mat board(800, 1000, CV_8UC3);

void drawPoint(Point pt, bool f)
{
    Scalar color;
    if (f == true)
        color = Scalar(0, 0, 255);
    else
        color = Scalar(255, 0, 0);

    circle(board, pt, 3, color, -1);
}
```

```
31.
32. DWORD WINAPI drawSquare()
33. {
34.     int w, h, l;
35.     P(mutex);
36.     w = board.cols;
37.     h = board.rows;
38.     V(mutex);
39.     l = min(w, h) * 0.8;
40.     int x = w / 2 - l / 2;
41.     int y = h / 2 - l / 2;
42.     double i = 0;
43.     while (i < l * 4)
44.     {
45.         int dx, dy;
46.         if (i < l)
47.         {
48.             dx = i;
49.             dy = 0;
50.         }
51.         else if (i < 2 * l)
52.         {
53.             dx = l;
54.             dy = i - l;
55.         }
56.         else if (i < 3 * l)
57.         {
58.             dx = 3 * l - i;
59.             dy = l;
60.         }
61.         else
62.         {
63.             dx = 0;
64.             dy = 4 * l - i;
65.         }
66.         Point p(x + dx, y + dy);
67.         P(mutex);
68.         drawPoint(p, 0);
69.         V(mutex);
70.         i += 4;
71.     }
72.     return 1;
73. }
74.
```

```
75. DWORD WINAPI drawCircle()
76. {
77.     double angle = 0;
78.     int x, y, r;
79.     P(mutex);
80.     x = board.cols / 2;
81.     y = board.rows / 2;
82.     V(mutex);
83.     r = min(x, y) * 0.8;
84.
85.     while (angle < 2 * PI)
86.     {
87.         int dx, dy;
88.         dx = r * cos(angle);
89.         dy = r * sin(angle);
90.         Point p(x + dx, y + dy);
91.         P(mutex);
92.         drawPoint(p, 1);
93.         V(mutex);
94.         angle += 0.01;
95.     }
96.
97.     return 1;
98. }
99.
100. int main()
101. {
102.     board = Scalar::all(255);
103.
104.     mutex = CreateSemaphore(NULL, 1, 1, NULL);
105.     CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)drawSquare, NULL, NULL, N
    ULL);
106.     CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)drawCircle, NULL, NULL, N
    ULL);
107.
108.     while (true)
109.     {
110.         P(mutex);
111.         imshow("Draw", board);
112.         char c = waitKey(10);
113.         if (c == 27)
114.             return 0;
115.         V(mutex);
116.     }
```

```
117.
118.     return 0;
119. }
```

# 模拟银行家算法

银行家算法：

银行家算法最初级原为银行系统设计，以确保银行在发放现金贷款时，不会发生不能满足所有客户需要的情况。在 OS 设计中，也可以用它来避免死锁。

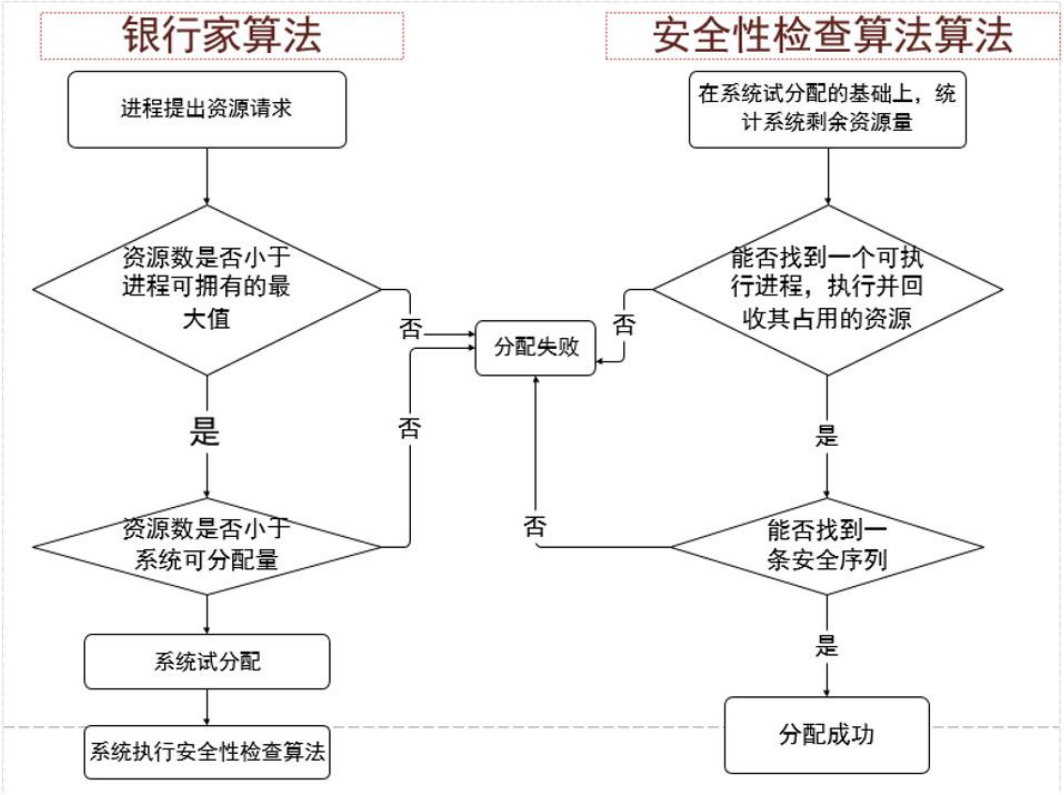
为实现银行家算法，每个新进程在进入系统时它必须申明在运行过程中，可能需要的每种资源类型的最大单元数目，其数目不应超过系统所拥有的资源总量。当某一进程请求时，系统会自动判断请求量是否小于进程最大所需，同时判断请求量是否小于当前系统资源剩余量。若两项均满足，则系统试分配资源并执行安全性检查算法。

安全性检查算法：

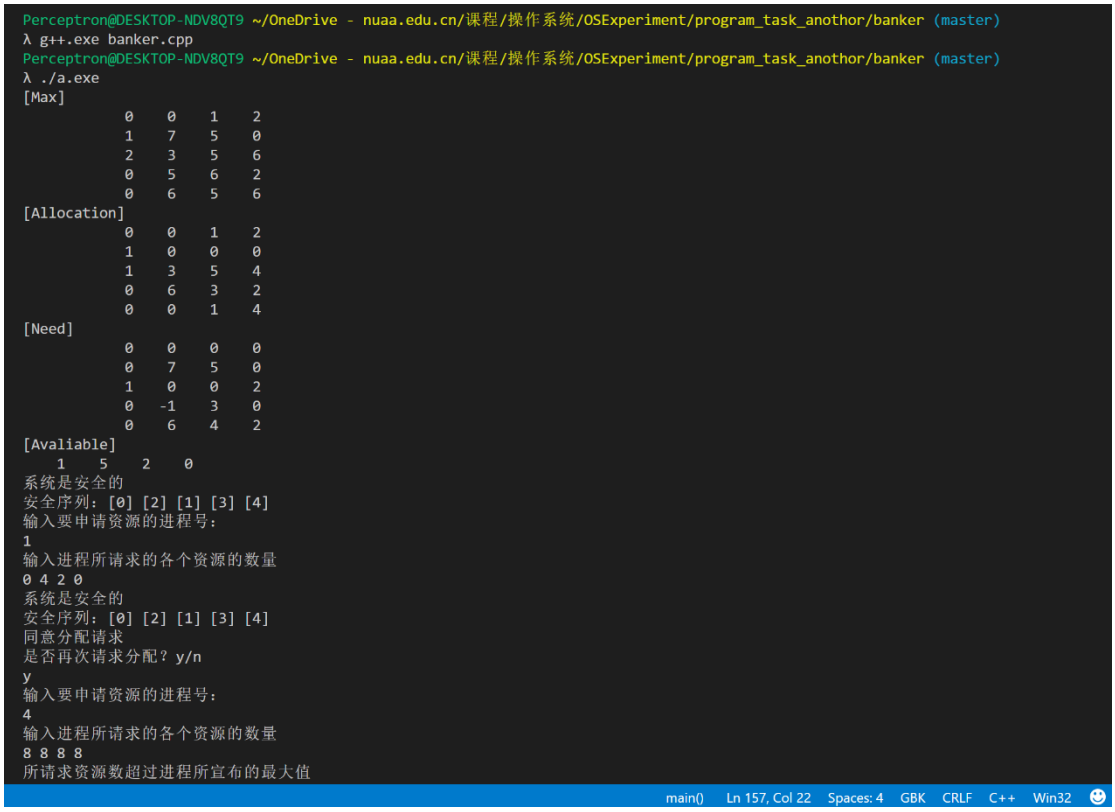
安全性检查算法用于检查系统进行资源分配后是否安全，若安全系统才可以执行此次分配；若不安全，则系统不执行此次分配。

安全性检查算法原理为：在系统试分配资源后，算法从现有进程列表寻找出一个可执行的进程进行执行，执行完成后回收进程占用资源；进而寻找下一个可执行进程。当进程需求量大于系统可分配量时，进程无法执行。当所有进程均可执行，则产生一个安全执行序列，系统资源分配成功。若进程无法全部执行，即无法找到一条安全序列，则说明系统在分配资源后会不安全，所以此次分配失败。

程序流程图：



实验截图：




代码：

```
#include <iostream>
#include <iomanip>
using namespace std;

//全局变量定义
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];    //M 个进程还需要 N 类资源的资源量
bool Finish[50];
int p[50];
int m, n;                //M 个进程，N 类资源

//安全性算法
int Safe()
{
    int i, j, k, l = 0;
    int Work[100];
    for(i = 0; i < n; i++)
        Work[i] = Available[i];
    for(i = 0; i < m; i++)
```

```
        Finish[i] = false;
    for(i = 0; i < m; i++)
    {
        if(Finish[i] == true)
            continue;
        for(j = 0; j < n; j++)
        {
            if(Need[i][j] > Work[j])
                break;
        }
        if(j != n)
            continue;   //不满足 Need < Work
        Finish[i] = true;
        for(k = 0; k < n; k++)
            Work[k] += Allocation[i][k];
        p[l++] = i;
        i = -1;
    }

    if(l == m)
    {
        cout << "系统是安全的" << endl;
        cout << "安全序列: ";
        for(i = 0; i < l; i++)
        {
            cout << "[" << p[i] << "] ";
        }
        cout << endl;
        return true;
    }
    else
    {
        cout << "安全序列不存在" << endl;
        return false;
    }
}

// Banker's Algorithm
int main()
{
    int i, j, mi;

    // 定义进程的数目和资源的种类
    m = 5;
```

```
67.     n = 4;
68.     // 定义最大需求矩阵
69.     int tmp[5][4] = {
70.         {0, 0, 1, 2},
71.         {1, 7, 5, 0},
72.         {2, 3, 5, 6},
73.         {0, 5, 6, 2},
74.         {0, 6, 5, 6}
75.     };
76.     // 定义分配矩阵
77.     int tmp2[5][4] = {
78.         {0, 0, 1, 2},
79.         {1, 0, 0, 0},
80.         {1, 3, 5, 4},
81.         {0, 6, 3, 2},
82.         {0, 0, 1, 4}
83.     };
84.     // 定义可利用资源向量
85.     int tmp3[4] = {1, 5, 2, 0};
86.
87.     for(i = 0; i < m; i++)
88.     {
89.         Available[i] = tmp3[i];
90.         for(j = 0; j < n; j++)
91.         {
92.             Max[i][j] = tmp[i][j];
93.             Allocation[i][j] = tmp2[i][j];
94.             Need[i][j] = Max[i][j] - Allocation[i][j];
95.         }
96.     }
97.
98.     // 输出当前状态
99.     cout << "[Max]" << endl;
100.     for(i = 0; i < m; i++)
101.     {
102.         cout << "\t";
103.         for(j = 0; j < n; j++)
104.             cout << setw(5) << Max[i][j];
105.         cout << endl;
106.     }
107.     cout << "[Allocation]" << endl;
108.     for(i = 0; i < m; i++)
109.     {
110.         cout << "\t";
```

```
111.         for(j = 0; j < n; j++)
112.             cout << setw(5) << Allocation[i][j];
113.         cout << endl;
114.     }
115.     cout << "[Need]" << endl;
116.     for(i = 0; i < m; i++)
117.     {
118.         cout << "\t";
119.         for(j = 0; j < n; j++)
120.             cout << setw(5) << Need[i][j];
121.         cout << endl;
122.     }
123.     cout << "[Avaliable]" << endl;
124.     for(j = 0; j < n; j++)
125.         cout << setw(5) << Available[j];
126.     cout << endl;
127.     // 判断当前系统状态是否安全
128.     if(!Safe())
129.     {
130.         cout << "系统不安全" << endl;
131.         return -1;
132.     }
133.     while(1)
134.     {
135.         cout << "输入要申请资源的进程号：" << endl;
136.         cin >> mi;
137.         cout << "输入进程所请求的各个资源的数量" << endl;
138.         for(i = 0; i < n; i++)
139.             cin >> Request[mi][i];
140.         bool flag = false;
141.         for(i = 0; i < n; i++)
142.         {
143.             if(Request[mi][i] > Need[mi][i])
144.             {
145.                 cout << "所请求资源数超过进程所宣布的最大值" << endl;
146.                 flag = true;
147.                 break;
148.             }
149.             if(Request[mi][i] > Available[i])
150.             {
151.                 cout << "无足够资源满足进程需求" << endl;
152.                 flag = true;
153.                 break;
154.             }
```

```
155.         }
156.         if(flag == true)
157.             continue;
158.         for(i = 0; i < n; i++)
159.         {
160.             Available[i] -= Request[mi][i];
161.             Allocation[mi][i] += Request[mi][i];
162.             Need[mi][i] -= Request[mi][i];
163.         }
164.         if(Safe())
165.         {
166.             cout << "同意分配请求" << endl;
167.         }
168.         else
169.         {
170.             cout << "拒绝分配请求" << endl;
171.             for(i = 0; i < n; i++)
172.             {
173.                 Available[i] += Request[mi][i];
174.                 Allocation[mi][i] -= Request[mi][i];
175.                 Need[mi][i] += Request[mi][i];
176.             }
177.         }
178.         char c;
179.         cout << "是否再次请求分配？y/n" << endl;
180.         while(1)
181.         {
182.             cin >> c;
183.             if(c == 'Y' || c == 'y')
184.             {
185.                 break;
186.             }
187.             else if(c == 'N' || c == 'n')
188.             {
189.                 cout << "Bye!" << endl;
190.                 return 0;
191.             }
192.             else
193.             {
194.                 cout << "请输入 y/n" << endl;
195.                 continue;
196.             }
197.         }
198.     }
```

```
199.
200.     return 0;
201. }
```

## 做一个动态链接并调用

操作系统：Windows 10 (64bit)
编译器：MinGW version 6.3.0
实验过程：

创建一个 c 文件，在里面编写一个 add 函数，然后用 gcc.exe 把这个 c 文件编译成动态链接库 dll；

创建一个 c 文件，在里面申明 add 函数，并写一个 main 函数来调用 add 函数，输出结果。

代码：

```
1.  // dlltest.c
2.
3.  int add(int x, int y)
4.  {
5.      return x + y;
6.  }
```

```
1.  // main.c
2.  #include <stdio.h>
3.
4.  int add(int x, int y);
5.
6.  int main()
7.  {
8.      int a = 89;
9.      int b = 11;
10.     int res = add(a, b);
11.     printf("%d + %d = %d\n", a, b, res);
12.     return 0;
13. }
```

实验截图：


Perceptron@DESKTOP-NDV8QT9 ~/OneDrive - nuaa.edu.cn/课程/操作系统/OSExperiment/program_task_anothor/dll (master)
λ ls
dlltest.c  main.c
Perceptron@DESKTOP-NDV8QT9 ~/OneDrive - nuaa.edu.cn/课程/操作系统/OSExperiment/program_task_anothor/dll (master)
λ gcc.exe dlltest.c -shared -o add.dll
Perceptron@DESKTOP-NDV8QT9 ~/OneDrive - nuaa.edu.cn/课程/操作系统/OSExperiment/program_task_anothor/dll (master)
λ ls
add.dll  dlltest.c  main.c
Perceptron@DESKTOP-NDV8QT9 ~/OneDrive - nuaa.edu.cn/课程/操作系统/OSExperiment/program_task_anothor/dll (master)
λ gcc.exe main.c -o test add.dll
Perceptron@DESKTOP-NDV8QT9 ~/OneDrive - nuaa.edu.cn/课程/操作系统/OSExperiment/program_task_anothor/dll (master)
λ ls
add.dll  dlltest.c  main.c  test.exe
Perceptron@DESKTOP-NDV8QT9 ~/OneDrive - nuaa.edu.cn/课程/操作系统/OSExperiment/program_task_anothor/dll (master)
λ ./test.exe
89 + 11 = 100
Perceptron@DESKTOP-NDV8QT9 ~/OneDrive - nuaa.edu.cn/课程/操作系统/OSExperiment/program_task_anothor/dll (master)
λ
(Global Scope) Ln 6, Col 2 Spaces: 4 UTF-8 CRLF C Win32