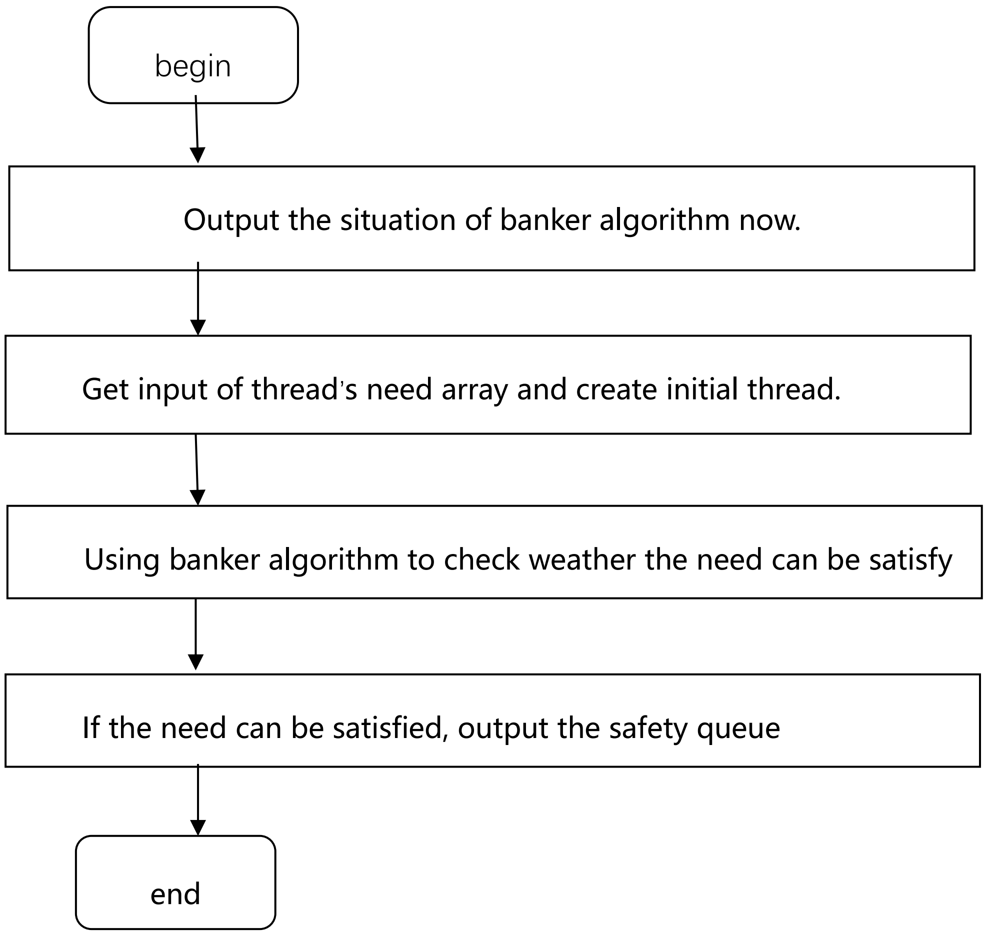
Operating system experiment report

1. **Name:** Luoyu Mei **Number:** 71117408 **Date:** 21/05/2019
2. **Working target:** Handle banker algorithm using Windows API, create multi thread to make satisfy critical section problem.

   Working environment: Window10 as basic operating system using G++ version 8.2.0 on Windows to handle banker algorithm.
3. **Steps:**
   1. Define available, allocation, max array, create basic banker algorithm.
   2. In order to make the program satisfy "Critical Section", I create five thread function respectively represent the application of system resource
   3. Write safety function using banker algorithm to check weather the application is safe, if safe then output safety queue, else reject the request.
   4. Get input of the need of process, using safety function to get the safety queue.
4. **Flow chart:**


begin
Output the situation of banker algorithm now.
Get input of thread's need array and create initial thread.
Using banker algorithm to check weather the need can be satisfy
If the need can be satisfied, output the safety queue
end

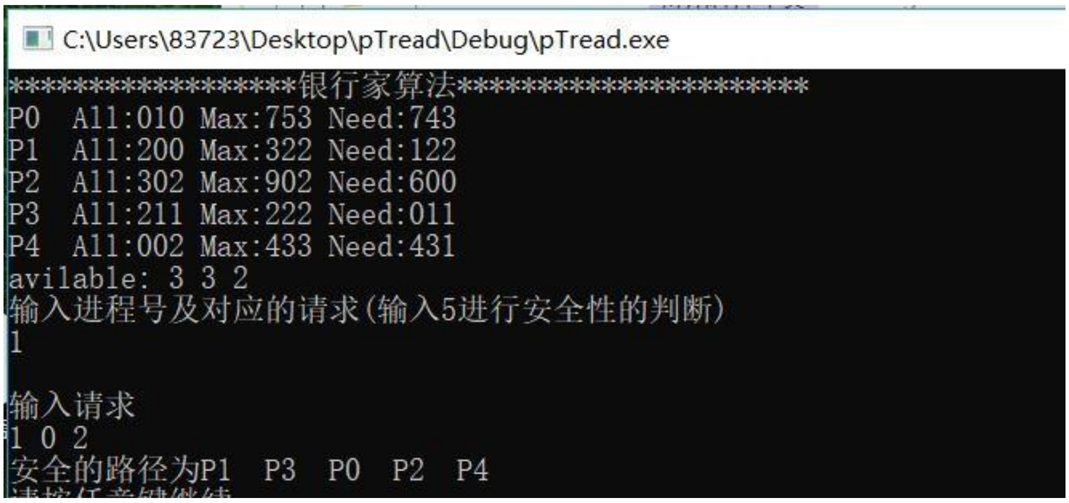
5. **Main data structure:**

I use five thread functions for five process, the available resource, allocation, max source of different process in system is saved using three arrays. After getting need queue form user, the program will firstly make sure it is safety using banker algorithm, when it is satisfy then output the safe queue.

The code file of my program will also be bale together with this report.

6. Experiment result:

Windows：



C:\Users\83723\Desktop\pTread\Debug\pTread.exe

```
*****************银行家算法*********************
P0  All:010 Max:753 Need:743
P1  All:200 Max:322 Need:122
P2  All:302 Max:902 Need:600
P3  All:211 Max:222 Need:011
P4  All:002 Max:433 Need:431
avilable: 3 3 2
输入进程号及对应的请求(输入5进行安全性的判断)
1

输入请求
1 0 2
安全的路径为P1  P3  P0  P2  P4
请按任意键继续
```

7. Filling of experiment:

It isn't so difficult for me to use Windows API to imply multi thread and make sure the critical section is accessed respectively. However, in order to use banker algorithm I need to write five function for five process and only change the allocation, available queue in system when the apply is safe.

8. Code for windows:

```cpp
#include <iostream>
#include <process.h>
#include <Windows.h>

using namespace std;
//定义 5 个线程函数，分别表示 5 个进程对系统资源的请求

DWORD WINAPI pro0(PVOID pvPram);

DWORD WINAPI pro1(PVOID pvPram);

DWORD WINAPI pro2(PVOID pvPram);

DWORD WINAPI pro3(PVOID pvPram);

DWORD WINAPI pro4(PVOID pvPram);
//申明安全函数

void ifsafe();

int available[3] = { 3, 3, 2 }; //系统当前可用的资源数
//5 个进程已分配的资源数

int alloc[5][3] = { {0, 1, 0},
                    {2, 0, 0},
                    {3, 0, 2},
                    {2, 1, 1},
                    {0, 0, 2}
};
//5 个进程的最大资源数

int max[5][3] = { {7, 5, 3},
                  {3, 2, 2},
                  {9, 0, 2},
                  {2, 2, 2},
                  {4, 3, 3}
};

int a, b, c, d; //用于用户的输入 b,c,d 表示进程的请求资源数

int main()
{
    //以一定的格式输出当前的资源分配情况
    cout << "*****************银行家算法*********************" << endl;
    cout << "P0 " << " All:" << alloc[0][0] << alloc[0][1] << alloc[0][2] << " Max:" << max[0][0] << max[0][1] << max[0][2] << " Need:" << max[0][0] - alloc[0][0] << max[0][1] - alloc[0][1] << max[0][2] - alloc[0][2] << endl;
    cout << "P1 " << " All:" << alloc[1][0] << alloc[1][1] << alloc[1][2] << " Max:" << max[1][0] << max[1][1] << max[1][2] << " Need:" << max[1][0] - alloc[1][0] << max[1][1] - alloc[1][1] << max[1][2] - alloc[1][2] << endl;
    cout << "P2 " << " All:" << alloc[2][0] << alloc[2][1] << alloc[2][2] << " Max:" << max[2][0] << max[2][1] << max[2][2] << " Need:" << max[2][0] - alloc[2][0] << max[2][1] - alloc[2][1] << max[2][2] - alloc[2][2] << endl;
    cout << "P3 " << " All:" << alloc[3][0] << alloc[3][1] << alloc[3][2] << " Max:" << max[3][0] << max[3][1] << max[3][2] << " Need:" << max[3][0] - alloc[3][0] << max[3][1] - alloc[3][1] << max[3][2] - alloc[3][2] << endl;
    cout << "P4 " << " All:" << alloc[4][0] << alloc[4][1] << alloc[4][2] << " Max:" << max[4][0] << max[4][1] << max[4][2] << " Need:" << max[4][0] - alloc[4][0] << max[4][1] - alloc[4][1] << max[4][2] - alloc[4][2] << endl;
    cout << "avilable: " << available[0] << " " << available[1] << " " << available[2] << endl;
    cout << "输入进程号及对应的请求(输入 5 进行安全性的判断)" << endl;
    cin >> a;
    cout << endl;
    if (a == 5)
    {
        ifsafe();
    }
    else
    {
        cout << "输入请求" << endl;
        cin >> b >> c >> d;
        //根据进程号来启用相应得进程
        switch (a)
        {
        case 0:
        {
            HANDLE pr0 = CreateThread(NULL, 0,
                pro0, NULL, 0,
                NULL);
            CloseHandle(pr0);
            break;
        }
        case 1:
        {
            HANDLE pr1 = CreateThread(NULL, 0,
                pro1, NULL, 0,
                NULL);
            CloseHandle(pr1);
            break;
        }
        case 2:
        {
            HANDLE pr2 = CreateThread(NULL, 0,
                pro2, NULL, 0,
                NULL);
            CloseHandle(pr2);
            break;
        }
        case 3:
        {
            HANDLE pr3 = CreateThread(NULL, 0,
                pro3, NULL, 0,
                    NULL);
                CloseHandle(pr3);
                break;
            }
            case 4:
            {
                HANDLE pr4 = CreateThread(NULL, 0,
                    pro4, NULL, 0,
                    NULL);
                CloseHandle(pr4);
                break;
            }
            }
    }
    system("pause");
    return 0;
}

DWORD WINAPI pro0(PVOID pvPram)
{
    //已分配的资源加上请求的资源数
    alloc[0][0] += b;
    alloc[0][1] += c;
    alloc[0][2] += d;
    //系统可用的资源数减去请求的资源数
    available[0] -= b;
    available[1] -= c;
    available[2] -= d;
    int x = 0;
    for (int i = 0; i < 3; i++)
    {
        if (available[i] < 0) //判断请求资源数是否大于系统可用的资源数
        {
            cout << "请求资源大于可用资源" << endl;
        }
        else
        {
            x++;
        }
    }
    if (x == 3)
        //如果三种类型的请求资源数都小于系统可用的资源数，则进行安全算法的判断
        ifsafe();
    return 0;
```

```cpp
}

DWORD WINAPI pro1(PVOID pvPram)
{
    alloc[1][0] += b;
    alloc[1][1] += c;
    alloc[1][2] += d;
    available[0] -= b;
    available[1] -= c;
    available[2] -= d;
    int x = 0;
    for (int i = 0; i < 3; i++)
    {
        if (available[i] < 0)
        {
            cout << "请求资源大于可用资源" << endl;
        }
        else
        {
            x++;
        }
    }
    if (x == 3)
        ifsafe();
    return 0;
}

DWORD WINAPI pro2(PVOID pvPram)
{
    alloc[2][0] += b;
    alloc[2][1] += c;
    alloc[2][2] += d;
    available[0] -= b;
    available[1] -= c;
    available[2] -= d;
    int x = 0;
    for (int i = 0; i < 3; i++)
    {
        if (available[i] < 0)
        {
            cout << "请求资源大于可用资源" << endl;
        }
        else
        {
            x++;
        }
    }
    if (x == 3)
        ifsafe();
    return 0;
}

DWORD WINAPI pro3(PVOID pvPram)
{
    alloc[3][0] += b;
    alloc[3][1] += c;
    alloc[3][2] += d;
    available[0] -= b;
    available[1] -= c;
    available[2] -= d;
    int x = 0;
    for (int i = 0; i < 3; i++)
    {
        if (available[i] < 0)
        {
            cout << "请求资源大于可用资源" << endl;
        }
        else
        {
            x++;
        }
    }
    if (x == 3)
        ifsafe();
    return 0;
}

DWORD WINAPI pro4(PVOID pvPram)
{
    alloc[4][0] += b;
    alloc[4][1] += c;
    alloc[4][2] += d;
    available[0] -= b;
    available[1] -= c;
    available[2] -= d;
    int x = 0;
    for (int i = 0; i < 3; i++)
    {
        if (available[i] < 0)
        {
            cout << "请求资源大于可用资源" << endl;
        }
        else
        {
            x++;
        }
    }
    if (x == 3)
        ifsafe();
    return 0;
}

void ifsafe()
{
    bool finish[5] = { false }; //每个进程当前的状态设为 false 表示该进程没有释放资源
    int n[5] = { 0 }; //记录数组，如果该进程 need 数小于 available，则将该进程号储存在数组中，最后用于输出
    int j = 0; //进行进程的遍历，初值为 0
    int q = 0; //计数器用于计数进程状态为 true 的个数
    int nn = 0; //
    for (; j < 5; j++)
    {
        int m = 0; //计数器，用于下面计数三种资源有几个满足要求
        if (finish[j] == true)
        {
            continue;
            //每次循环前要判断该进程是否已经完成了资源的释放，如果已经释放，则跳过
        }
        else
        {
            for (int i = 0; i < 3; i++)
            {
                if ((max[j][i] - alloc[j][i]) <= available[i]) //需求的资源小于系统可用的资源数
                {
                    m++;
                }
            }
            if (m == 3) //当 m=3 说明系统可以满足该进程的需求
            {
                n[nn] = j; //记录进程号
                finish[j] = true; //释放该进程
                for (int k = 0; k < 3; k++)
                {
                    available[k] += alloc[j][k]; //系统可用资源的回收
                }
                j = -1; //将 j=-1 是为了下一次进行进程的遍历时，要从头开始，防止前面的一些进程当时不满足要求但现在可以了
                nn++;
                //记录数组下一次记录要从下一个位子开始
            }
        }
    }
    for (int i = 0; i < 5; i++)
    {
        if (finish[i] == false)
        {
            cout << "不存在安全算法" << endl;
            //只要 5 个进程有一个没有通过而释放资源，说明当前系统不安全
        }
        else
        {
            q++;
        }
    }
    if (q == 5) //如果 q=5 说明系统所以进程都释放了资源，系统安全
    {
        cout << "安全的路径为";
        for (int i = 0; i < 5; i++)
        {
            cout << "P" << n[i] << "  ";      //一个一个释放记录数组中记录的进程号
        }
        cout << endl;
    }
}
```