实验序号：01　　实验项目名称：进程同步与互斥

学号	8208190406	姓名	王瑛	专业班级	计科 1902 班
实验地点	新校区	指导教师	沈海澜	时间	2021.5.23

一、　实验编号

实验一 进程同步与互斥

二、　实验目的

1、掌握基本进程（线程）同步与互斥算法，理解生产者-消费者模型。

2、学习使用 windows 或 Linux 平台下进程/线程创建、终止以及同步控制相关的 API 函数使用方法。

3、设计 window 或 Linux 下应用程序，实现生产者-消费者进程（线程）的同步与互斥。

三、　实验环境

Windows 10 + Visual Studio 2019

四、　实验内容

以生产者-消费者问题为依据，在 windows 或 linux 环境下创建一个控制台进程，在该进程中创建 n 个生产者和消费者，实现线程的同步和互斥。

五、　实验分析与设计

分析：①在主函数中创建多个生产者线程和消费者线程，线程随机进行访问

②当一个线程进入临界区时，其他线程访问想要访问该临界区时将会被挂起，一直等到该线程释放临界区

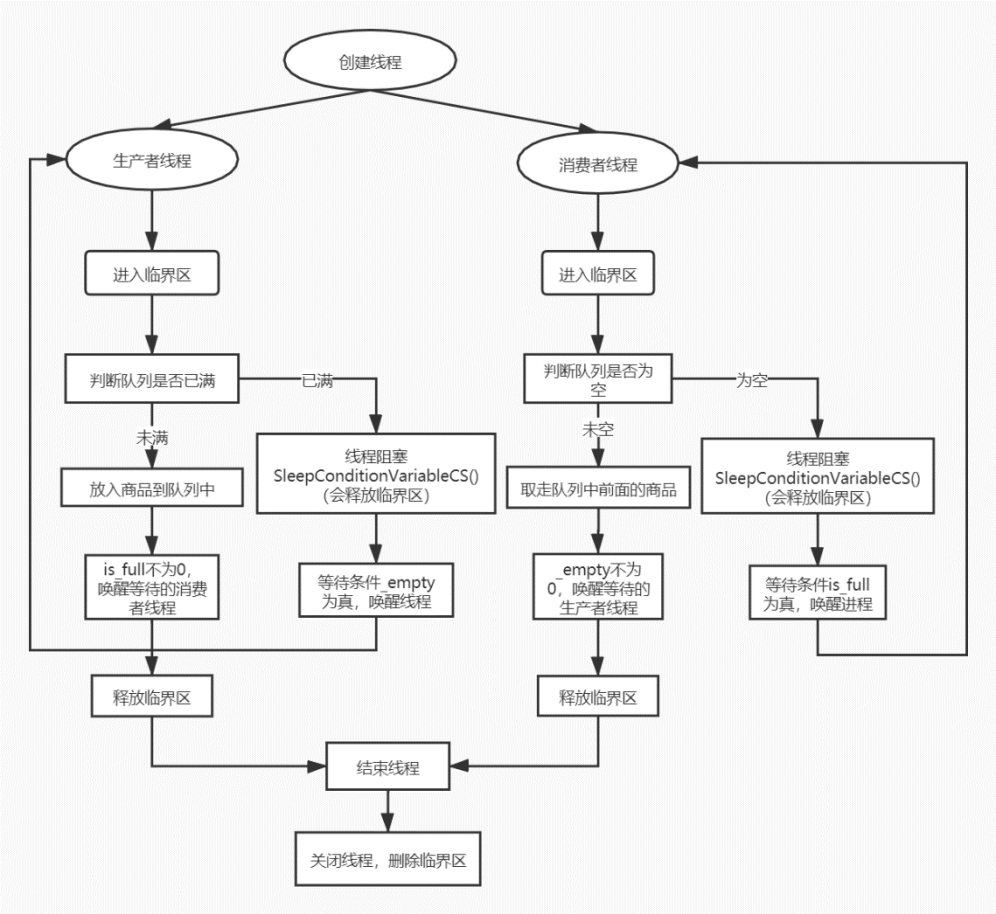
③生产者放入商品之前要考虑商品队列是否已满，如果已经满了的话，该生产者线程将会被阻塞，释放临界区进入休眠，等待它的条件变量_empty 变为真后被唤醒

④消费者在取商品之前要判断商品队列是否为空，如果为空的话，该消费者线程将会被阻塞，释放临界区进入休眠，等待它的条件变量 is_full 变为真后被唤醒

⑤当按下 ctr+Z 后结束程序运行，生产者退出，消费者取完队列中的商品后退出

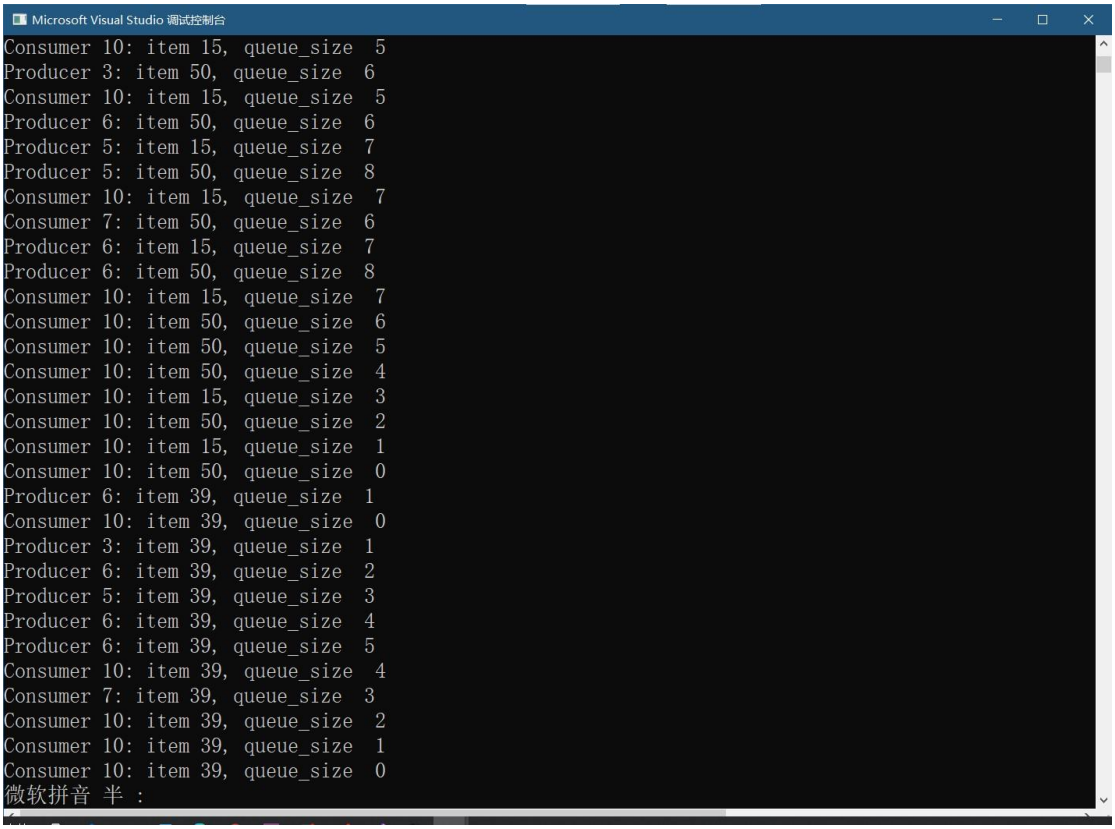
⑥线程终止运行之后，关闭线程对象，删除临界区结构对象

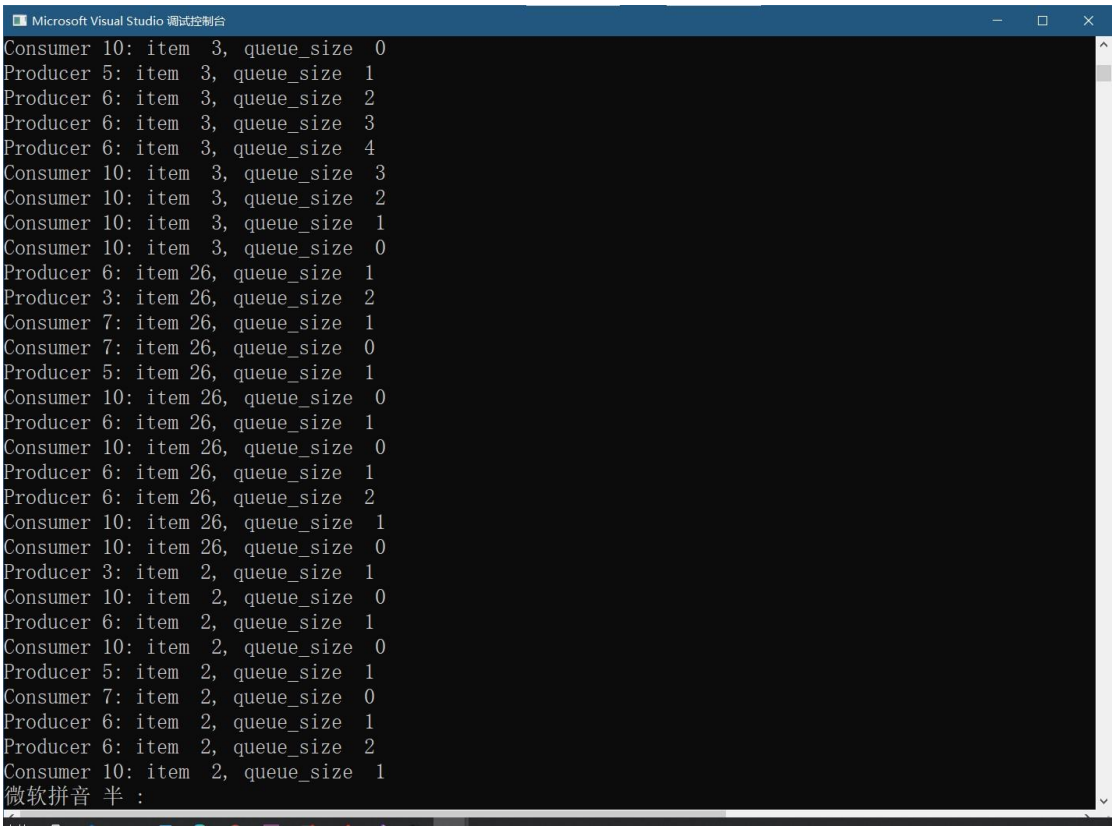
算法流程图：

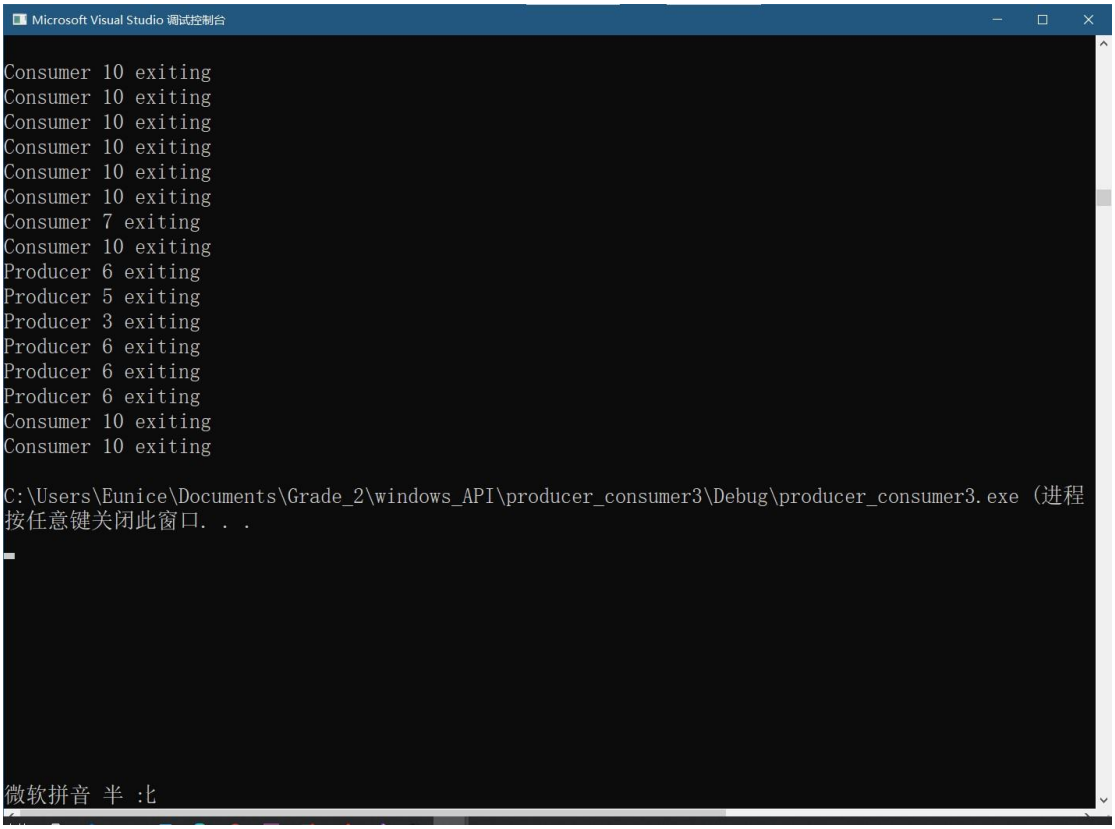


设置的数据结构：生产者和消费者线程设置了顺序表的形式，商品设置队列空间来进行存、取，作为资源空间判断空和满

六、　实验运行结果与相关代码


Microsoft Visual Studio 调试控制台
Consumer 10: item 15, queue_size 5
Producer 3: item 50, queue_size 6
Consumer 10: item 15, queue_size 5
Producer 6: item 50, queue_size 6
Producer 5: item 15, queue_size 7
Producer 5: item 50, queue_size 8
Consumer 10: item 15, queue_size 7
Consumer 7: item 50, queue_size 6
Producer 6: item 15, queue_size 7
Producer 6: item 50, queue_size 8
Consumer 10: item 15, queue_size 7
Consumer 10: item 50, queue_size 6
Consumer 10: item 50, queue_size 5
Consumer 10: item 50, queue_size 4
Consumer 10: item 15, queue_size 3
Consumer 10: item 50, queue_size 2
Consumer 10: item 15, queue_size 1
Consumer 10: item 50, queue_size 0
Producer 6: item 39, queue_size 1
Consumer 10: item 39, queue_size 0
Producer 3: item 39, queue_size 1
Producer 6: item 39, queue_size 2
Producer 5: item 39, queue_size 3
Producer 6: item 39, queue_size 4
Producer 6: item 39, queue_size 5
Consumer 10: item 39, queue_size 4
Consumer 7: item 39, queue_size 3
Consumer 10: item 39, queue_size 2
Consumer 10: item 39, queue_size 1
Consumer 10: item 39, queue_size 0
微软拼音 半 ：


Microsoft Visual Studio 调试控制台
Consumer 10: item 3, queue_size 0
Producer 5: item 3, queue_size 1
Producer 6: item 3, queue_size 2
Producer 6: item 3, queue_size 3
Producer 6: item 3, queue_size 4
Consumer 10: item 3, queue_size 3
Consumer 10: item 3, queue_size 2
Consumer 10: item 3, queue_size 1
Consumer 10: item 3, queue_size 0
Producer 6: item 26, queue_size 1
Producer 3: item 26, queue_size 2
Consumer 7: item 26, queue_size 1
Consumer 7: item 26, queue_size 0
Producer 5: item 26, queue_size 1
Consumer 10: item 26, queue_size 0
Producer 6: item 26, queue_size 1
Consumer 10: item 26, queue_size 0
Producer 6: item 26, queue_size 1
Producer 6: item 26, queue_size 2
Consumer 10: item 26, queue_size 1
Consumer 10: item 26, queue_size 0
Producer 3: item 2, queue_size 1
Consumer 10: item 2, queue_size 0
Producer 6: item 2, queue_size 1
Consumer 10: item 2, queue_size 0
Producer 5: item 2, queue_size 1
Consumer 7: item 2, queue_size 0
Producer 6: item 2, queue_size 1
Producer 6: item 2, queue_size 2
Consumer 10: item 2, queue_size 1
微软拼音 半 ：



代码：

```
#include <iostream>
#include "stdio.h"
#include "stdlib.h"
#include "time.h"
#include "windows.h"
#define Max_size 10
#define NUM_PRODUCER 6
#define NUM_CONSUMER 10
#define PRODUCER_SLEEP_TIME_MS 500
#define CONSUMER_SLEEP_TIME_MS 500
using namespace std;

long int buffer[Max_size];
int k;
int t;
long TotalItemsProduced; //生产的商品总数
long TotalItemsConsumed; //消费的商品总数
BOOL StopRequested;  //停止请求
CRITICAL_SECTION CS; //临界区
CONDITION_VARIABLE _empty; //线程休眠和唤醒的条件变量，下同（在vs里面is_empty会报错，所以这里命名就改成了_empty
CONDITION_VARIABLE is_full;
DWORD WINAPI Producer(LPVOID); //生产者线程
```

```
DWORD WINAPI Consumer(LPVOID); //消费者线程

int main() {
    int i;
    HANDLE h1[NUM_PRODUCER], h2[NUM_CONSUMER]; //10个生产者10个消费者

    InitializeCriticalSection(&CS); //初始化临界区
    InitializeConditionVariable(&_empty); //初始化条件变量，下同
    InitializeConditionVariable(&is_full);
    for (int i = 0; i < NUM_PRODUCER; i++) { //创建生产者线程
        h1[i] = CreateThread(NULL, 0, &Producer, &i, 0, NULL);
    }
    for (int i = 0; i < NUM_CONSUMER; i++) { //创建消费者线程，中间没有设置路障，这些线程运行顺序是随机的
        h2[i] = CreateThread(NULL, 0, &Consumer, &i, 0, NULL);
    }

    puts("Press enter to stop...");
    getchar(); //同时按住ctrl+Z停止运行

    EnterCriticalSection(&CS);
    StopRequested = TRUE;
    LeaveCriticalSection(&CS);

    WakeAllConditionVariable(&_empty); //唤醒在这个条件变量上休眠的所有线程，下同
    WakeAllConditionVariable(&is_full);
    WaitForMultipleObjects(NUM_PRODUCER, h1, true, INFINITE);//主线程等待这一组线程完成，第一个是让函数查看的内核对象的数量，第二个是指向内核对象句柄的指针
    //第三个传true，在所有对象变为已通知之前，该函数不允许调用线程运行，第四个是一个超时值，等待时间到了，函数无论如何都会返回
    WaitForMultipleObjects(NUM_CONSUMER, h2, true, INFINITE);
    for (int i = 0; i < NUM_PRODUCER; i++) {
        CloseHandle(h1[i]);  //线程终止运行后，线程对象仍在系统中，通过CloseHandle()函数来关闭该线程对象
    }
    for (int i = 0; i < NUM_CONSUMER; i++) {
        CloseHandle(h2[i]);
    }
    DeleteCriticalSection(&CS); //对临界区使用完之后调用该函数删除临界区结构的对象
    return 0;
}

DWORD WINAPI Producer(LPVOID p) {
    int ThreadNum = *(int*)p;
```

```
    while (true) {
        // Produce a new item.
        int item;
        EnterCriticalSection(&CS); //进入临界区，其他要访问临界区的线程将被挂起要一直等到临界区被释放
        item = (rand() % 80) + 1; //随机生成商品的值
        while (k == Max_size && StopRequested == FALSE) { //物品队列满了，生产者线程阻塞
            // Buffer is full - sleep, so consumers can get items.
            SleepConditionVariableCS(&_empty, &CS, INFINITE);  //线程休眠，第一个线程等待被唤醒的条件变量
        }
        if (StopRequested == TRUE) //请求停止
        {
            LeaveCriticalSection(&CS); //释放临界区
            break;
        }
        // Insert the item at the end of the queue and increment size.
        buffer[(k + t)%Max_size] = item; //商品进入物品队列中
        k++; //当前队列的长度加一
        TotalItemsProduced++;  //总生产量加一
        printf("Producer %u: item %2d, queue_size %2u\r\n", ThreadNum, item, k);

        LeaveCriticalSection(&CS);  //释放临界区
        WakeConditionVariable(&is_full);  // If a consumer is waiting, wake it.

        Sleep(rand() % PRODUCER_SLEEP_TIME_MS);
    }

    printf("Producer %u exiting\r\n", ThreadNum);
    return 0;
}

DWORD WINAPI Consumer(LPVOID p) {
    int ThreadNum = *(int*)p;
    while (true) {

        int item;
        EnterCriticalSection(&CS);
        while (k == 0 && StopRequested == FALSE) {
            // Buffer is empty - sleep so producers can create items.

            SleepConditionVariableCS(&is_full, &CS, INFINITE);
        }
```

```
        if (StopRequested == TRUE && k == 0)
        {

            LeaveCriticalSection(&CS);
            break;
        }
        // Consume the first available item.
        item = buffer[t];
        k--; //队列长度减一
        t++;
        TotalItemsConsumed++;
        if (t == Max_size)
        {
            t = 0;
        }
        printf("Consumer %u: item %2d, queue_size %2u\r\n", ThreadNum, item, k);

        LeaveCriticalSection(&CS);//释放临界区
        WakeConditionVariable(&_empty);// If a producer is waiting, wake it.
        Sleep(rand() % CONSUMER_SLEEP_TIME_MS);
    }
    printf("Consumer %u exiting\r\n", ThreadNum);
    return 0;
}
```

七、 遇到的问题及解决办法

① **问题一**：多个生产者和多个消费者随机访问：

一开始的时候设置的是一个生产者生产完之后一个消费者取商品，每次都设置 WaitForSingleObject(semaphore,INFINITE)控制顺序，这样就没有实现线程随机地访问

解决办法：不设置路障，设置数组，线程的运行是随机的：

```
for (int i = 0; i < NUM_PRODUCER; i++) { //创建生产者线程
        h1[i] = CreateThread(NULL, 0, &Producer, &i, 0, NULL);
}
for (int i = 0; i < NUM_CONSUMER; i++) { //创建消费者线程
        h2[i] = CreateThread(NULL, 0, &Consumer, &i, 0, NULL);
 }
```

② **问题二**：当线程要运行的条件不满足时，要进入阻塞状态：

解决办法：设置条件变量进行控制：

```
CONDITION_VARIABLE _empty; //线程休眠和唤醒的条件变量，下同
CONDITION_VARIABLE is_full;
while (k == Max_size && StopRequested == FALSE) {
   // Buffer is full - sleep, so consumers can get items.
   SleepConditionVariableCS(&_empty, &CS, INFINITE);
}
while (k == 0 && StopRequested == FALSE) {
   // Buffer is empty - sleep so producers can create items.

   SleepConditionVariableCS(&is_full, &CS, INFINITE);
}
```

③ **问题三：**唤醒线程

解决办法：当生产者放入商品之后，队列就不为空，这时若有在等待的消费者线程，则将其唤醒；当消费者取走商品之后，这是队列就不是满的，这时若有在等待的生产者线程，则将其唤醒：

```
WakeConditionVariable(&is_full);  // If a consumer is waiting, wake it.
WakeConditionVariable(&_empty);// If a producer is waiting, wake it.
```

④ **问题四：**最后不要忘记关闭线程对象和删除临界区结构对象

解决办法：调用 CloseHandle()和 DeleteCriticalSection()：

```
for (int i = 0; i < NUM_PRODUCER; i++) {
  CloseHandle(h1[i]);  //线程终止运行后，线程对象仍在系统中，通过CloseHandle()函数来关闭该线程对象
}
for (int i = 0; i < NUM_CONSUMER; i++) {
  CloseHandle(h2[i]);
}
  DeleteCriticalSection(&CS); //对临界区使用完之后调用该函数删除临界区结构对象
```