实验二　生产者消费者问题

班级：07112004　学号：1120201883　姓名：刘博文

一、实验目的

学习生产者与消费者的运行基本原理，学习使用共享内存区和信号量机制，学习使用多进程以及进程间通信的方法，学会使用锁互斥访问对象。

二、实验内容

- 创建一个有 6 个缓冲区的缓冲池，初始为空，每个缓冲区能存放一个长度若为 10 个字符的字符串。
- 2 个生产者进程
  - 随机等待一段时间，往缓冲区添加数据，
  - 若缓冲区已满，等待消费者取走数据后再添加
  - 重复 12 次
- 3 个消费者进程
  - 随机等待一段时间，从缓冲区读取数据
  - 若缓冲区为空，等待生产者添加数据后再读取
  - 重复 8 次

说明：

- 在 Windows 平台上做。
- 显示每次添加或读取数据的时间及缓冲区的映像。
- 生产者和消费者用进程模拟。

三、程序设计与实现

根据实验要求我们知道至少要完成以下概念的代码实现：缓冲池、生产者、消费者；同时我们还要规定相关的信号量。先给出相关宏变量定义：

```
#define COUNT_BUFFER 6          //6 个缓冲区
#define BUFFER_LENGTH 10        //每个缓冲区数组最低长度为 10
#define COUNT_PRODUCER 2        //2 个生产者进程
#define REPEAT_PRODUCER 12      //重复 12 次
#define COUNT_CONSUMER 3        //3 个消费者进程
```

```
#define REPEAT_CONSUMER 8        //重复 8 次
#define COUNT_PROCESS 5          //总进程数
```

信号量

信号量 empty 表示为空的缓冲区数，初值为 6

信号量 full 表示为满的缓冲区数，初值为 0

信号量 mutex 用于读写互斥，初值为 1

缓冲池

使用一个指针数组来保存指向六个缓冲区的指针

对于一个缓冲区，其要能容纳长度为 10 个字符的字符串，那么它的大小至少要是 10 个字节；在这里可以用长度为 10 的 char 类型的数组来模拟；那么 6 个缓冲区组成的缓冲池可以用长度为 60 的一维数组模拟（这里尝试过用二维数组模拟，但是在运行时会出莫名其妙的 bug 而用一维数组则不会）

不仅如此，根据题意，我们还要知道缓冲区是否是空的，所以要设定两个指针，一个指针指向数组最后一个有数据的空间，这两个指针和数组自带的数组头部指针即可判断数组为空或为满

具体数据结构如下所示：

```
struct BufferPool{
   char bufferZones[COUNT_BUFFER * BUFFER_LENGTH]; //存储具体数据
   char * bufferZoneStart[COUNT_BUFFER];          //存储每个缓冲区的起始指针
   char * bufferZoneEnd[COUNT_BUFFER];            //存储每个缓冲区的尾指针，默认状态下与起始指针相同，在写入内容后变化
}BufferPool;
```

生产者

生产者至少要完成以下任务：

```
//申请空闲缓冲区
P(empty);
//申请修改缓冲区的权限
P(mutex);
write();
```

```
//释放填充缓冲区信号量
V(Full);
//释放缓冲区修改权限
V(mutex);
```

消费者

消费者至少要完成以下任务：

```
//申请已填充缓冲区资源
P(full);
//申请缓冲区修改权限
P(mutex);
read();
//释放空缓冲区资源
V(empty);
//释放修改权限
V(mutex);
```

主进程

主进程要负责创建公共信号量、缓冲区与缓冲池、2 个生产者进程和 3 个消费者进程

具体函数作用

1. CreateSharedMemory()

   这个函数用于在主进程创建 BufferPool 类型共享内存区，并对该内存区格式化

2. 信号量相关函数

   CreateMux()、OpenMux()、CloesMux()分别负责相关信号量的创建、打开和关闭

3. CreateSubProcess(int ID)

   接收一个 ID，然后创建一个子进程

4. Producer(int ID)

   生产者进程所运行的函数

5. Consumer(int ID)

消费者进程运行的函数

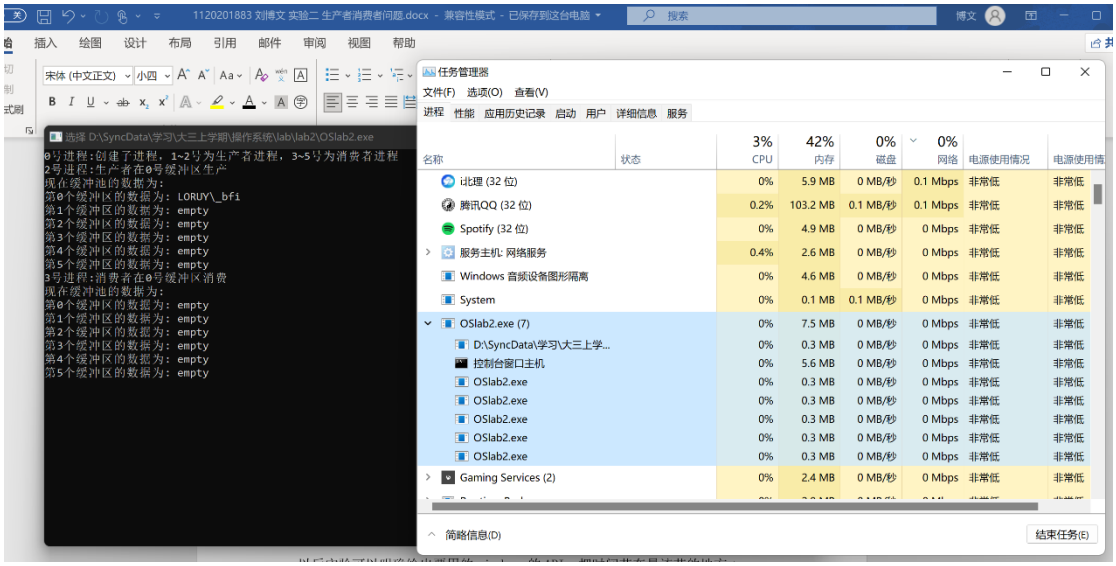
6. main 函数

负责创建共享内存区、信号量、子进程和回收句柄和信号量等资源

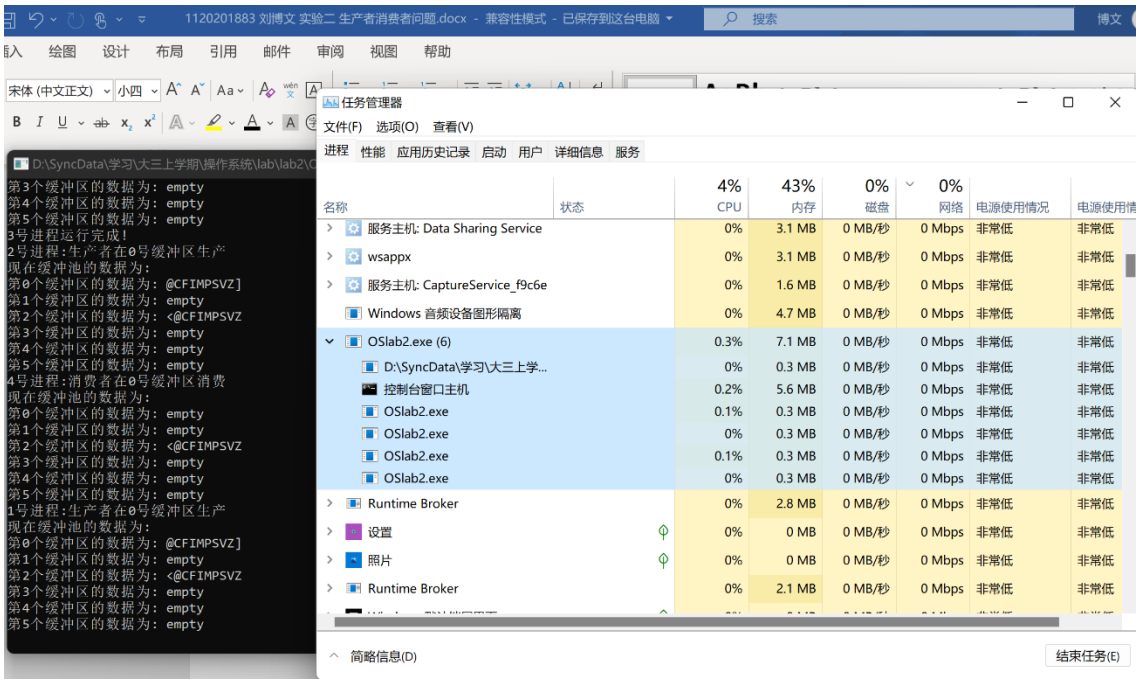
所有的源码在随报告附带的 main.c 中，里面有比较详细的注释可供参考

四、实验结果及分析

双击 OSlab2.exe（随报告附带）运行，任务管理器中可以看到除了命令行进程之外有 6 个进程，其中有 5 个子进程：



而当程序运行一段时间之后有进程退出时也可以清晰的在任务管理器中看到进程的减少：



而且根据命令行中打印的信息也可以知道程序的各个进程在正确的运行，实验成功完成。

五、实验收获与体会

这个实验原理上不是很难，导致大部分时间用来学习 windows 的 API，希望以后实验可以明确给出要用的 windows 的 API，把时间花在最该花的地方。