操作系统课程实验报告

学生姓名：/

班 学 号：/

指导教师：/

地理与信息工程学院

2019 年 5 月 28 日

实习题目：基于 Windows 进程互斥实现机制

【需求规格说明】

基于 Microsoft Visual Studio 环境的多线程编程验证互斥的原理，理解多线程编程中关键变量的定义与使用，通过使用 Semaphore，mutex 等控制变量，实现对生产者消费者模型的真实模拟。函数的功能与实现自己定义。

【算法设计】

（1） **设计思想：**

与书上的伪码设计思想一样，因为书上的代码已经写的很清楚了，我无非是对其的扩充与相关函数的实现。当缓冲区 buffer 为空时，等待生产者生产，消费者停止消费；当缓冲区 buffer 为满时，等待消费者消费，生产者停止生产。

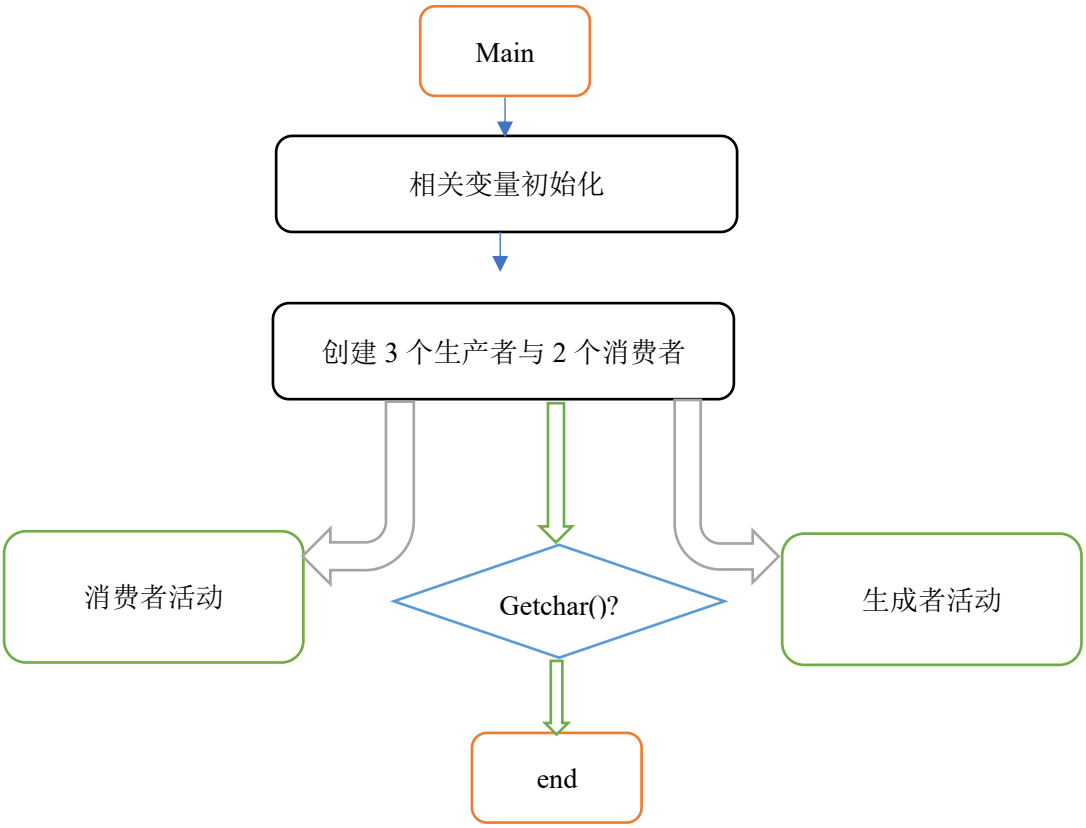
设计一个 Mymutex 控制对临界区的访问，Myfull 纪录缓冲区满的槽数，控制消费者的行为，Myempty 纪录缓冲区空的槽数，控制生产者的行为。

此处我设计了 3 个生产者与 2 个消费者，会出现进程等待的情况。

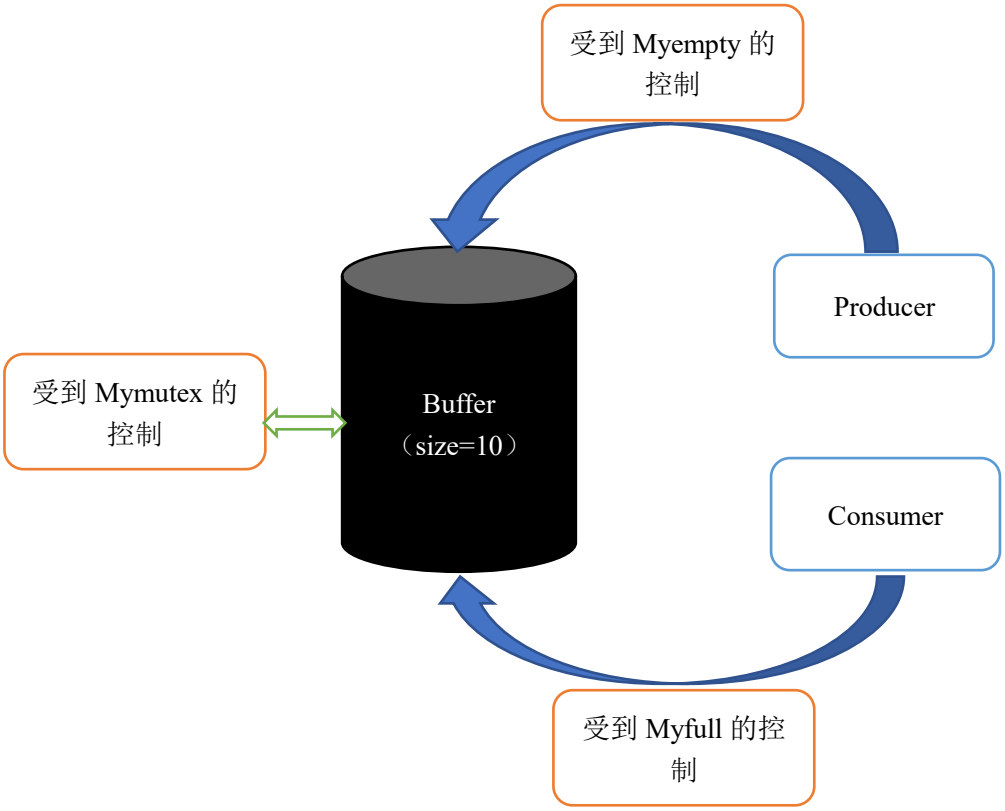
缓冲区的实现：一个大小为 10 的数组，循环消费与添加。

产生 item：简单的 item++。

（2） **详细设计：**



消费者与生产者：



【调试报告】

1） 刚开始设置主线程等待子线程结束，则无法手动结束；

2） 生成 item 用 rand()发现很多重复，弃用，改为简单的自加；

Microsoft Visual Studio 调试控制台

```
create producer 0
create producer 1
create producer 2
create consumer 0
create consumer 1
produce 1 in buffer 0
produce 3 in buffer 1
produce 4 in buffer 2
consume 1 in buffer 0
produce 6 in buffer 3
produce 7 in buffer 4
consume 3 in buffer 1
produce 8 in buffer 5
consume 4 in buffer 2
produce 9 in buffer 6
produce 10 in buffer 7
consume 6 in buffer 3
produce 11 in buffer 8
consume 7 in buffer 4
produce 12 in buffer 9
produce 13 in buffer 0
consume 8 in buffer 5
produce 14 in buffer 1
consume 9 in buffer 6
produce 15 in buffer 2
produce 16 in buffer 3
consume 10 in buffer 7


E:\2019上\课件\操作系统原理\实习\practice1\Debug\practice1.exe (进程 11396)已退出，返回代码为: 0。
若要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口...
```

【附录】

Producer：

```
unsigned __stdcall producer(HANDLE) {
    //int item;
    while (run)
    {
        //item = produce_item();
        item = item + 1;
        WaitForSingleObject(Myempty, INFINITE);
        WaitForSingleObject(Mymutex, INFINITE);
        insert_item(item);
        Sleep(500);
        ReleaseMutex(Mymutex);
        ReleaseSemaphore(Myfull, 1, NULL);
    }
    return 0;
}
```

Consumer：

```
unsigned __stdcall consumer(HANDLE) {
    while (run)
    {
        WaitForSingleObject(Myfull, INFINITE);
        WaitForSingleObject(Mymutex, INFINITE);
        remove_item();
        Sleep(500);
        ReleaseMutex(Mymutex);
        ReleaseSemaphore(Myempty, 1, NULL);
    }
    return 0;
}
```