

实验名称	生产者消费者问题		
学号	1120200563	姓名	肖良寿
一、实验目的 1. 了解生产者消费者问题的互斥与同步问题 (1). 理解互斥的基本定义与实现原理 (2). 理解同步的基本原理及其与互斥之间的关系 2. 掌握 Windows 系统下多进程编程的方法 (1). 了解进程的创建方法 (2). 熟悉 Windows 系统下进程间通信的实现 二、实验内容 1. 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. 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. 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 三、实验环境及配置方法 1. Windows11-64 位操作系统 2. Visual Studio 2022 社区版 集成开发环境 四、实验方法和实验步骤（程序设计与实现）			

1. 问题分析

(1) **关系分析** 生产者与消费者对缓冲区的访问是互斥关系,同时生产者与消费者存在相互协作的关系——只有生产者生产后消费者才能消费,二者存在同步关系。

(2) **整理思路** 只有生产者与消费者两种进程,存在同步与互斥关系,需要解决的问题为互斥与同步的 PV 操作的位置

(3) **信号量设置** 信号量 mutex 为互斥信号量,控制互斥访问缓冲池,初始化为 1; 信号量 full 记录当前缓冲池中的“满”缓冲区数量,初始化为 0; 信号量 empty 记录当前缓冲池中的“空”缓冲区数量,初始化为 n。

2. 伪代码表示

```
Semaphore mutex = 1;
Semaphore empty = n; //空闲缓冲区
Semaphore full = 0; //缓冲区初始化为空

Producer(){ //生产者进程
    For(i=1 to 12){
        Produce an item in nextp ;
        P(empty); //申请空缓冲区单元
        P(mutex); //进入临界区
        Add nextp to buffer;
        //向临界区写数据
        V(mutex);
        //离开临界区,释放互斥信号量
        V(full);
        //满缓冲区数量+1
    }
}
```

```
Consumer(){
    For(i=1 to 8){
        P(full); //申请缓冲区单元
        P(mutex); //进入临界区
        Remove an item from buffer;
        //向缓冲区取走数据
        V(mutex);
        //离开临界区,释放互斥信号量
        V(empty);
        //空缓冲区数量+1
        Consume the item;
    }
}
```

3. C++程序设计

(1) 信号量的设置可调用 C++ 语言的内置函数 `CreatMutex()`、`OpenSemaphore()` 来实现；

(2) P、V 操作调用 C++ 语言的内置函数 `WaitForSingleObject()`、`ReleaseMutex()`、`ReleaseSemaphore()` 函数来实现；

(3) 多进程创建

Windows 创建多进程使用 `CreatProcess()` 函数调用自己，通过多次创建得到 2 个生产者进程和 3 个消费者进程，在之中运行相应的生产者进程函数与消费者进程函数。在通过传入参数不同，来辨别是第一次主进程还是生产者进程，消费者进程。

```

TCHAR szFilename[MAX_PATH];
GetModuleFileName(NULL, szFilename, MAX_PATH);

TCHAR szCmdLine[MAX_PATH];
sprintf(szCmdLine, "\\\"%s\\\" %d", szFilename, nCloneID);

STARTUPINFO si;
ZeroMemory(reinterpret_cast<void*>(&si), sizeof(si));

si.cb = sizeof(si);
PROCESS_INFORMATION pi;

BOOL bCreateOK = CreateProcess(
    szFilename,
    szCmdLine,
    NULL,
    NULL,
    FALSE,
    CREATE_DEFAULT_ERROR_MODE,
    NULL,
    NULL,
    &si,
    &pi);
if (bCreateOK)
    handleOfProcess[nCloneID] = pi.hProcess;
else
{
    printf("Error in create process!\n");
    exit(0);
}

```

(4) 进程间通信

通过构建共享内存区实现进程间通信，首先调用 `OpenFileMapping()` 创建一个共享内存空间并返回一个 HANDLE 句柄，此后调用 `MapViewOfFile()` 获取内存映射到该程序的内存（本进程）；

五、实验结果和分析

程序运行的控制台结果如下图所示：

```
Microsoft Visual Studio 调试器
生产者9248向缓冲区写入数据: Z Z @16:06:11.438
生产者22624向缓冲区写入数据: Z Z @16:06:11.438
生产者22624向缓冲区写入数据: Z J @16:06:12.478
生产者9248向缓冲区写入数据: J @16:06:12.478
消费者11408从缓冲区读取数据: J @16:06:12.478
消费者22564从缓冲区读取数据: J @16:06:12.478
消费者11408从缓冲区读取数据: @16:06:13.523
生产者22624向缓冲区写入数据: @16:06:13.523
生产者9248向缓冲区写入数据: @16:06:13.523
消费者13968从缓冲区读取数据: @16:06:13.523
消费者11408从缓冲区读取数据: @16:06:14.571
生产者9248向缓冲区写入数据: @16:06:14.571
生产者22624向缓冲区写入数据: @16:06:14.571
消费者22564从缓冲区读取数据: @16:06:14.571
消费者11408从缓冲区读取数据: @16:06:15.619
消费者13968从缓冲区读取数据: @16:06:15.619
生产者9248向缓冲区写入数据: @16:06:15.619
生产者22624向缓冲区写入数据: @16:06:15.619
消费者22564从缓冲区读取数据: @16:06:16.671
消费者11408从缓冲区读取数据: @16:06:16.671
生产者22624向缓冲区写入数据: R @16:06:16.671
生产者9248向缓冲区写入数据: R R @16:06:16.671
消费者13968从缓冲区读取数据: T @16:06:17.722
消费者22564从缓冲区读取数据: T @16:06:17.722
生产者22624向缓冲区写入数据: T @16:06:17.722
生产者9248向缓冲区写入数据: T @16:06:17.722
生产者22624向缓冲区写入数据: T @16:06:18.774
生产者9248向缓冲区写入数据: T @16:06:18.774
消费者11408从缓冲区读取数据: @16:06:18.774
消费者13968从缓冲区读取数据: @16:06:18.774
生产者22624向缓冲区写入数据: @16:06:19.833
消费者11408从缓冲区读取数据: @16:06:19.833
生产者9248向缓冲区写入数据: @16:06:19.833
消费者22564从缓冲区读取数据: @16:06:19.833
消费者13968从缓冲区读取数据: @16:06:20.893
消费者11408从缓冲区读取数据: @16:06:20.893
生产者9248向缓冲区写入数据: @16:06:20.893
生产者22624向缓冲区写入数据: @16:06:20.893
消费者13968从缓冲区读取数据: @16:06:21.966
生产者9248向缓冲区写入数据: B @16:06:21.966
消费者22564从缓冲区读取数据: @16:06:21.966
生产者22624向缓冲区写入数据: B B @16:06:21.966
生产者22624向缓冲区写入数据: L @16:06:23.43
消费者13968从缓冲区读取数据: L @16:06:23.43
生产者9248向缓冲区写入数据: L @16:06:23.43
消费者22564从缓冲区读取数据: B L @16:06:23.43
消费者13968从缓冲区读取数据: @16:06:24.115
消费者22564从缓冲区读取数据: @16:06:24.115

E:\ProgrammingFiles\C++\OS\x64\Debug\OS.exe (进程 21904)已退出，代码为 0。
按任意键关闭此窗口。 . . |
```

六、讨论、心得

程序的运行过程并不直观，只能看到程序的运行结果，调用、跳转关系还比较黑盒，如能设计更加可视化的程序，对于理解操作系统进程、进程间通信等将会有很好的效果。

