**操作系统 Project2**

*13331231*

*孙圣*

*教务4班（补修）*

1. **实验环境与使用**

操作系统：Windows8

使用说明：打开cmd，进入指定目录。

1.由于使用了<psapi.h>头文件，编译时需要设置linker，即添加编译属性 -lpsapi；

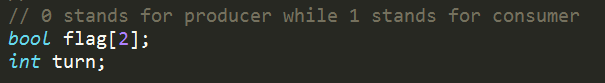
2.批处理测试：执行execute.bat；

3.一般测试：执行pc.exe [#producers] [#consumers] [timeToSleep] [mode]，即生产者线程数量，消费者线程数量，执行时间和模式。例如 pc.exe 1 1 10 2，即创建一个生产者和一个消费者进程，在模式2(使用互斥锁和信号量)下执行10秒。

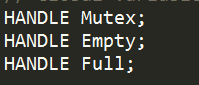
共有3种模式：

模式0：没有做任何的保护，即counter、in/out等全局变量可能会被同时改变，因此只是作为一个对照；

模式1：使用了Peterson’ solution，即利用flag和turn来表示该轮到哪个进程执行：



模式2：使用了Mutex Lock和Semaphore，其中有1个互斥锁用来限定只有一个进程进入临界区，有2个信号量用来记录buffer是否为空或满：



1. **实验过程**

先判断命令行参数的个数，如果不为4个，则直接退出。

之后判断各个参数的范围，例如producers 和consumers的个数都必须在1-10个之间，执行时间(timeToSleep)必须在1-60秒之间，而模式(mode)为0-2。

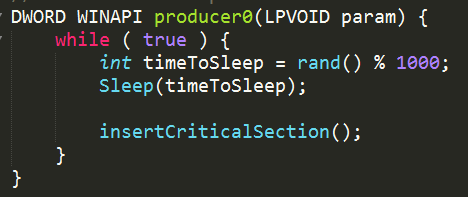
在执行之前，需要将所有的全局变量初始化。程序中将其封装在了init()函数中，这包括：buffer中元素的数量、buffer的in/out指针位置，还有用于统计生产者消费者操作数目的变量。

还要定义一些辅助函数，例如用于插入和删除buffer中的insert()和remove()函数。当成功操作时，返回0；否则返回1。

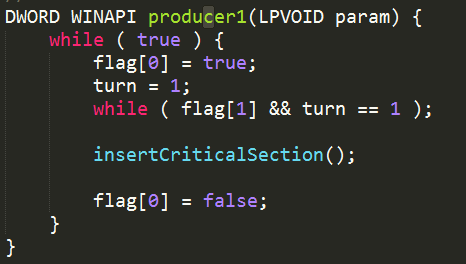
之后再定义critical section中的插入和删除函数，此处定义为inline，为了减少函数调用中的开销。每进行一次操作，totalOp的数量加一；当成功操作时,realOp的数量加一：

然后就是根据不同的模式来创建相应的线程：

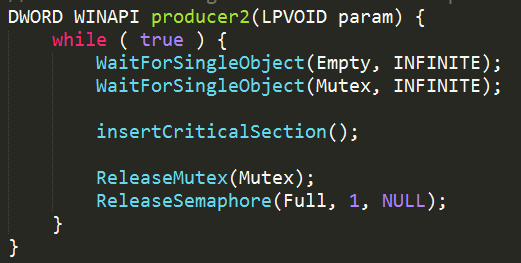
对于模式0，不需要太多的操作，只需要在执行操作前，随机选择一定的时间进行休眠(0-999ms)，之后执行相应的临界区操作即可：



对于模式1，生产者进程在进入临界区之前要将自己的flag设置为true，同时把turn设置为1。如果消费者进程已经在临界区中执行时，则一直在while处不断循环等待、直到条件不满足而跳出循环，进入临界区。完成了相应的操作后，将自己的flag置为false，表示自己暂时并不需要再执行临界区中的代码：



对于模式2，当buffer不为满的时候，生产者进程才能够进行操作，因此要对Empty信号量进行P操作，这里利用到了Win32 API中的WaitForSingleObject()方法，之后还要对Mutex互斥锁进行P操作，保证消费者进程不会同时进入临界区。完成相应的临界区操作之后，生产者分别对Mutex和Full信号量进行V操作，这里分别利用了ReleaseMutex()和ReleaseSemaphore()方法。要注意的是，生产者等待的是Empty信号量，而signal的是Full信号量，因为当其向buffer中插入元素时，buffer的大小会加一，而这刚好与Full的定义相同：



线程创建的顺序也是遵循随机的原则：先随机产生值为0或1的数，当值为0，且生产者还没有创建完毕时，创建一个生产者；同理，当值为1，且消费者还没有创建完毕时，创建一个消费者。当全都创建完毕时，跳出循环，主线程自动休眠所设定的时间，让各个线程自由执行。

最后，程序输出相应的性能信息，这又包括了两个方面，一方面是时间信息，另一方面是内存开销：

对于时间效率的衡量主要是通过单位时间内所执行的操作的数目来判断的，即同一时间内执行的操作越多，则说明方法的时间效率越高。

因此只要把统计的四个操作数目逐一输出即可，分别是：

producerRealOp 生产者实际的操作数(因为存在buffer为满，无法插入的情况)

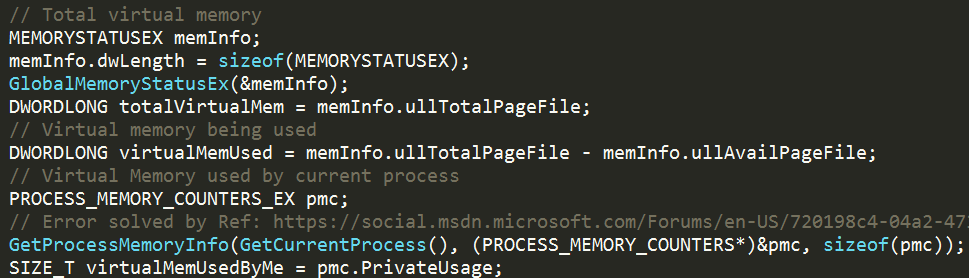
consumerRealOp 消费者实际的操作数(因为存在buffer为空，无法移除的情况)

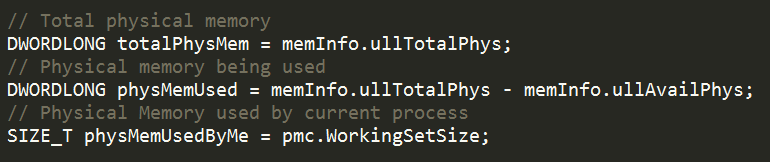
producerTotalOp 生产者总的操作数

consumerTotalOp 消费者总的操作数

对于空间开销的衡量，主要是利用了Win32 API中提供的MEMORYSTATUSEX来实现，一共统计了6个参数，分别是：虚拟内存的大小，正在使用的虚拟内存的大小，被该进程使用的虚拟内存的大小，物理内存的大小，正在使用的物理内存的大小和被该进程使用的物理内存的大小。

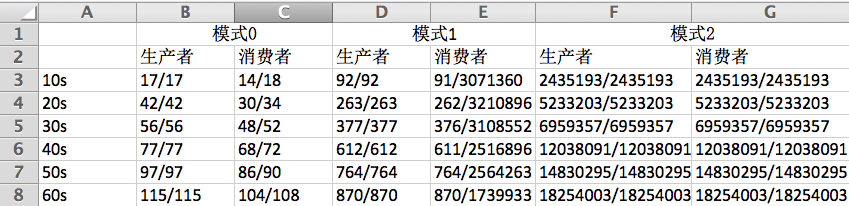
将以上信息转换为KB后输出，并计算了相应的内存使用率：



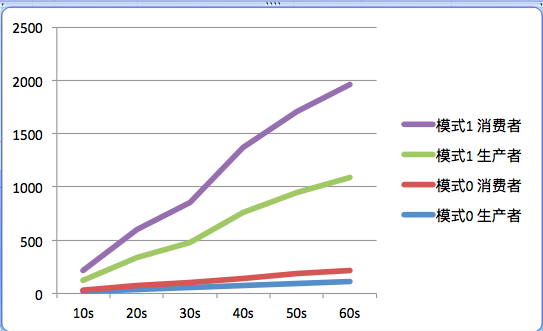


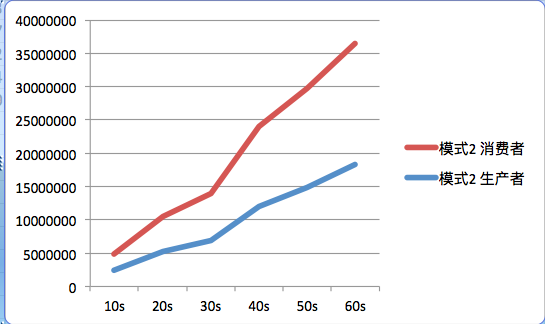
1. **实验结果**

1. 1个生产者线程和1个消费者线程运算数目比较，执行时间从10s到60s（左侧为实际成功进行的操作的数目，右侧为总共尝试的操作的数目）



增长趋势图：

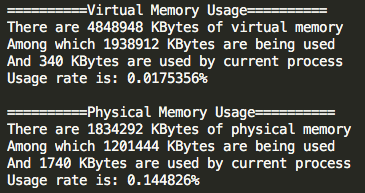




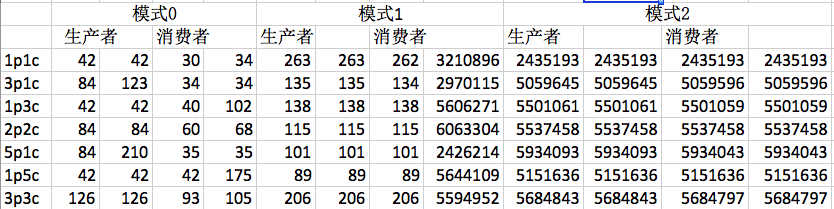
分析：对于模式0来说，所尝试和执行的操作的数目都是最少的，因为在尝试每个操作之前都要随机休眠0-999毫秒，造成了时间的浪费，因此效率相对低下；对于模式1来说，由于采用了turn和flag而避免了休眠的时间，因此实际执行操作数目与模式0相比有大约5-8倍的提升，而对于消费者，所尝试的操作数量也大幅度提升至20-30万左右；对于模式2来说，由于利用了信号量和互斥锁等操作系统提供的方法来解决问题，因此效率非常高，甚至达到了千万的级别。综上，模式2的效率是最高的，模式1次之，模式0的效率最低。

对于内存的使用：三种方法都占用了大约340KB的虚拟内存和1740KB的物理内存，因此差别不大。

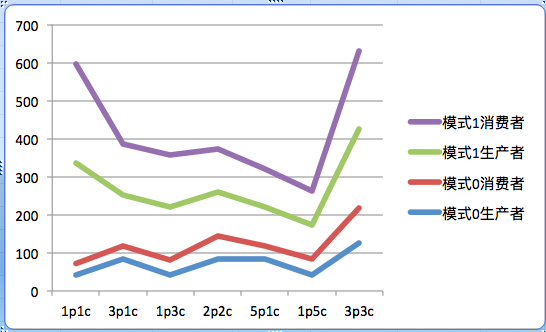
同时可以发现一个特点：程序在执行的时候占用虚拟内存的比例是比较小的，而占用的内存相对较大，约为15%：

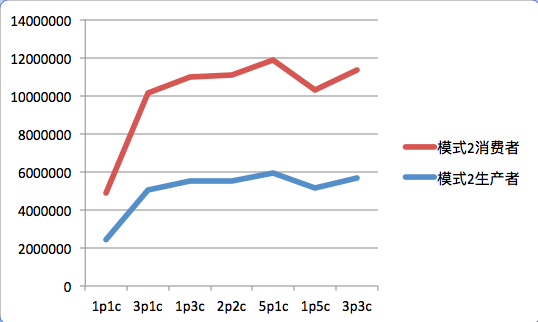


2.多个生产者线程和多个消费者线程运算数目比较，执行时间20秒，其中p代表生产者，c代表消费者（左侧为实际成功进行的操作的数目，右侧为总共尝试的操作的数目）



趋势图：





分析：根据图可知，模式0和模式1的趋势基本相同，但与模式2完全不一样。对于模式0，如果只看生产者数目和消费者数目等同的情况，可以发现由1p1c到2p2c再到3p3c，执行的操作的数目是逐渐递增的；而对于模式1来说，2p2c的情况执行的操作数目是最少的，线段呈一个抛物线，具体原因不太清楚；而对于模式2来说，2p2c的情况和3p3c的情况基本相同，这说明在2p2c的时候已经达到一个瓶颈，因此3p3c时提高不大。

对于1p3c,3p1c,2p2c这三种情况，由于线程数都为4个，因此执行的操作数也基本相同。

参考资料：

[1] 查看内存开销

<http://stackoverflow.com/questions/63166/how-to-determine-cpu-and-memory-consumption-from-inside-a-process>

[2] 解决GetProcessMemoryInfo()方法中的错误

<https://social.msdn.microsoft.com/Forums/en-US/720198c4-04a2-4737-9159-6e23a217d6b7/question-about-getprocessmemoryinfo?forum=Vsexpressvc>