操作系统 Project2

13331231
孙圣
教务 4 班（补修）

一、 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：

```
// 0 stands for producer while 1 stands for consumer
bool flag[2];
int turn;
```

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：

```
HANDLE Mutex;
HANDLE Empty;
HANDLE Full;
```

二、 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：

```
DWORD WINAPI producer0(LPVOID param) {
    while ( true ) {
        int timeToSleep = rand() % 1000;
        Sleep(timeToSleep);

        insertCriticalSection();
    }
}
```

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：

```
DWORD WINAPI producer1(LPVOID param) {
    while ( true ) {
        flag[0] = true;
        turn = 1;
        while ( flag[1] && turn == 1 );

        insertCriticalSection();

        flag[0] = false;
    }
}
```

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：

```
DWORD WINAPI producer2(LPVOID param) {
    while ( true ) {
        WaitForSingleObject(Empty, INFINITE);
        WaitForSingleObject(Mutex, INFINITE);

        insertCriticalSection();

        ReleaseMutex(Mutex);
        ReleaseSemaphore(Full, 1, NULL);
    }
}
```

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：

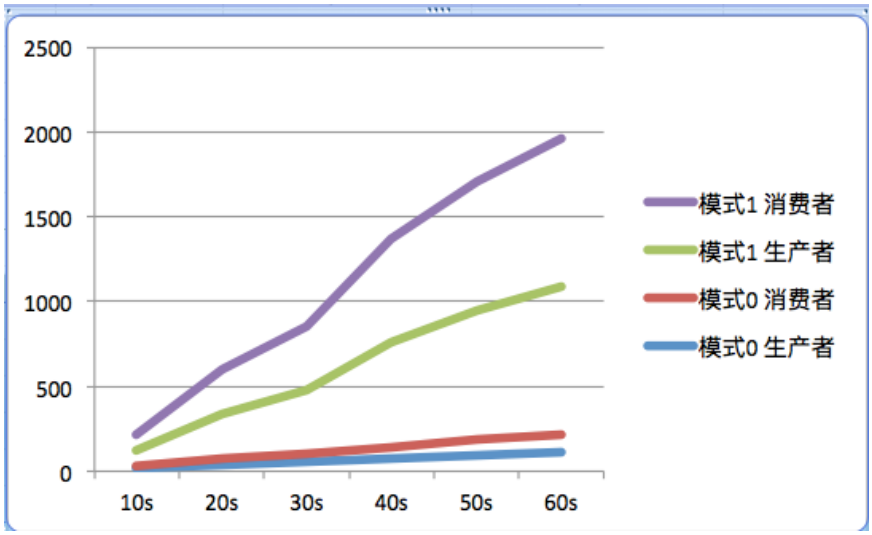
```
// Total virtual memory
MEMORYSTATUSEX memInfo;
memInfo.dwLength = sizeof(MEMORYSTATUSEX);
GlobalMemoryStatusEx(&memInfo);
DWORDLONG totalVirtualMem = memInfo.ullTotalPageFile;
// Virtual memory being used
DWORDLONG virtualMemUsed = memInfo.ullTotalPageFile - memInfo.ullAvailPageFile;
// Virtual Memory used by current process
PROCESS_MEMORY_COUNTERS_EX pmc;
// Error solved by Ref: https://social.msdn.microsoft.com/Forums/en-US/720198c4-04a2-47
GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS*)&pmc, sizeof(pmc));
SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;
// Total physical memory
DWORDLONG totalPhysMem = memInfo.ullTotalPhys;
// Physical memory being used
DWORDLONG physMemUsed = memInfo.ullTotalPhys - memInfo.ullAvailPhys;
// Physical Memory used by current process
SIZE_T physMemUsedByMe = pmc.WorkingSetSize;
```

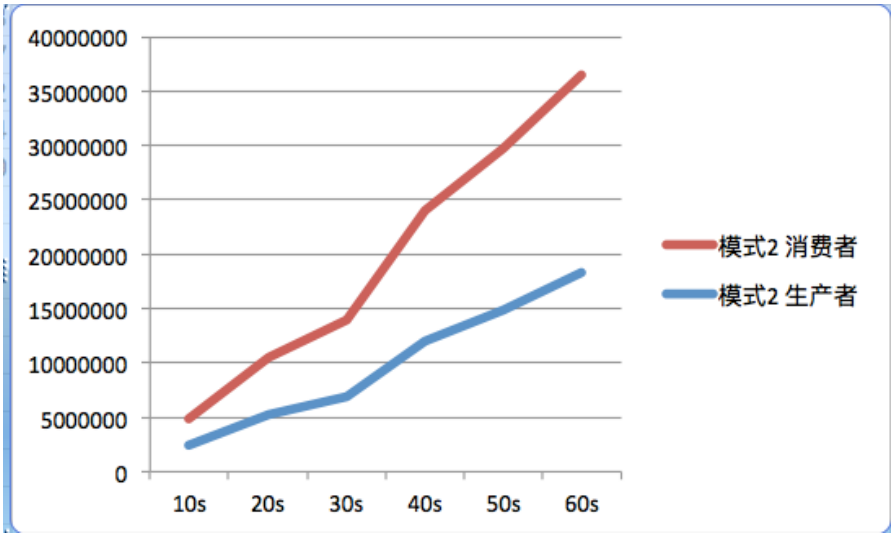
三、 实验结果

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）

	A	B	C	D	E	F	G
1		模式0		模式1		模式2	
2		生产者	消费者	生产者	消费者	生产者	消费者
3	10s	17/17	14/18	92/92	91/3071360	2435193/2435193	2435193/2435193
4	20s	42/42	30/34	263/263	262/3210896	5233203/5233203	5233203/5233203
5	30s	56/56	48/52	377/377	376/3108552	6959357/6959357	6959357/6959357
6	40s	77/77	68/72	612/612	611/2516896	12038091/12038091	12038091/12038091
7	50s	97/97	86/90	764/764	764/2564263	14830295/14830295	14830295/14830295
8	60s	115/115	104/108	870/870	870/1739933	18254003/18254003	18254003/18254003

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。

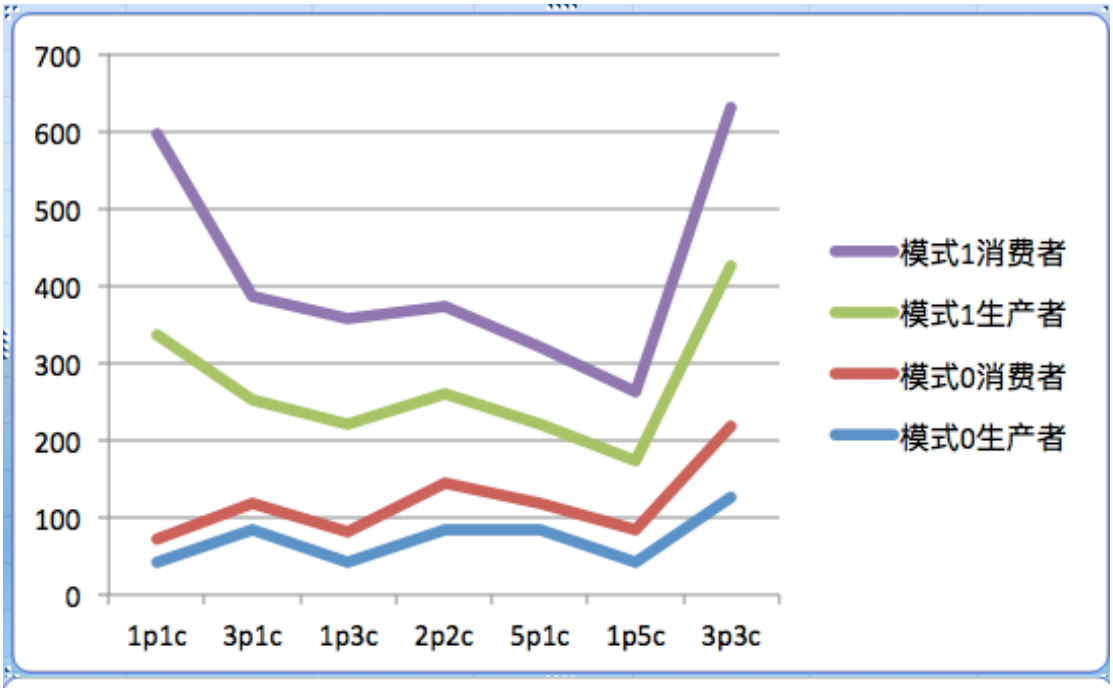
同时可以发现一个特点：程序在执行的时候占用虚拟内存的比例是比较小的，而占用的内存相对较大，约为 15%：

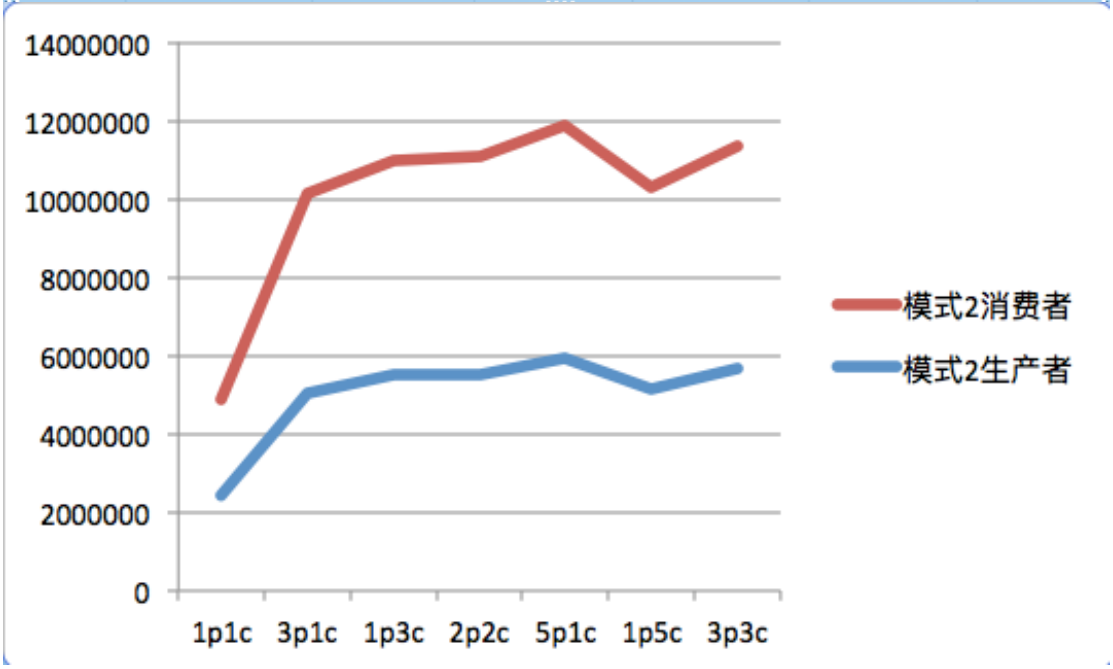
```
==========Virtual Memory Usage==========
There are 4848948 KBytes of virtual memory
Among which 1938912 KBytes are being used
And 340 KBytes are used by current process
Usage rate is: 0.0175356%

==========Physical Memory Usage==========
There are 1834292 KBytes of physical memory
Among which 1201444 KBytes are being used
And 1740 KBytes are used by current process
Usage rate is: 0.144826%
```

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）

	模式0				模式1				模式2			
	生产者		消费者		生产者		消费者		生产者		消费者	
1p1c	42	42	30	34	263	263	262	3210896	2435193	2435193	2435193	2435193
3p1c	84	123	34	34	135	135	134	2970115	5059645	5059645	5059596	5059596
1p3c	42	42	40	102	138	138	138	5606271	5501061	5501061	5501059	5501059
2p2c	84	84	60	68	115	115	115	6063304	5537458	5537458	5537458	5537458
5p1c	84	210	35	35	101	101	101	2426214	5934093	5934093	5934043	5934043
1p5c	42	42	42	175	89	89	89	5644109	5151636	5151636	5151636	5151636
3p3c	126	126	93	105	206	206	206	5594952	5684843	5684843	5684797	5684797

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