OS Labs:

生产者消费者（PC）问题

问题描述：

实现 I 个生产者 J 个消费者问题，其中共享缓冲区的大小为 N，所有生产者共生产 K(K>N)个产品后结束，所有消费者共消费 K 个产品后结束。

具体要求：

1) 严格按时序输出每个生产者、消费者的行为，其中包括生产产品 k、消费产品 k、进入临界区、存入产品、取出产品、离开临界区；
2) 需要考虑边界（某生产者生产第 K 个产品后所有生产者结束；某消费者消费第 K 个产品后所有消费者结束）
3) 需要考虑随机函数，生产者生产时需要一个随机时间；消费者消费时也需要一个随机时间；
4) 编号：无论生产者还是消费者都需要有编号；产品同样也需要编号；缓冲区的各个产品项也需要有编号；
5) 输出形式可以采用标准输出、图形动态显示及同时文字记录输出等方式，无论是生产者还是消费者，其主要输出内容如下：
   a) 进入临界区前，输出某某编号（生产者/消费者）线程准备进入临界区
   b) 进入临界区后，输出某某编号（生产者/消费者）线程已进入临界区
   c) 离开临界区后，输出某某编号（生产者/消费者）线程已离开临界区
   d) 生产者生产一个产品时，需要输出产品信息；
   e) 生产者将产品放入缓冲区时，需要输出相关信息；
   f) 消费者将产品从缓冲区取出时，需要输出相关信息；
   g) 消费者消费一个产品时，需要输出产品信息；
6) *不能出现竞态
7) **不能出现忙等待

参考结构：

```
#define DATA_SIZE 256
/*产品结构体声明*/
typedef struct item_st {
    int         id              /*产品编号*/
    time_t      timestamp_p;  /*生产时间*/
    int         produce_id;     /生产者编号*/
    DWORD checksum; /*  产品校验码*/
    DWORD data[DATA_SIZE]; /*  产品详细信息*/
} ITEM_ST;

/*缓冲区结构体声明*/
typedef struct buffer_st {
    int             id;             /*缓冲区编号*/
    boolean         isempty;        /*是否为空*/
    ITEM_ST         item;           /*具体产品*/
    buffer_st*      nextbuf;        /*下一个缓冲区首地址*/
} BUFFER_ST;
```

实现平台：

1） Windows32 API
2） Java
3） POSIX C （GNU/Linux）

Windows 技术参考：

1） 线程技术 **CreateThread** 或 _beginthreadex
   a) CreateThread
   b) GetCurrentThreadId
   c) GetThreadPriority
   d) SetThreadPriority
   e) Sleep
   f) SuspendThread
   g) ResumeThread
   h) ExitThread

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES lpThreadAttributes, // SD
  DWORD dwStackSize,                        // initial stack size
  LPTHREAD_START_ROUTINE lpStartAddress, // thread function
  LPVOID lpParameter,                       // thread argument
  DWORD dwCreationFlags,                    // creation option
  LPDWORD lpThreadId                        // thread identifier
);
```

2） 线程同步
   a) WaitForSingleObject
   b) WaitForMultipleObjects
3） 随机数
   a) srand ((DWORD)time(NULL)); /* Seed the random # generator */
   b) rand()   /* generate a random */

c) Sleep() /*msec*/

4） 临界区 CRITICAL_SECTION
a) InitializeCriticalSection
b) DeleteCriticalSection
c) EnterCriticalSection
d) LeaveCriticalSection

5） 事件 event
a) CreateEvent
b) SetEvent
c) ResetEvent
d) PulseEvent

6） 互斥对象
a) CreateMutex
b) （OpenMutex）
c) WaitForSingleObject(hMutex, INFINITE)
d) ReleaseMutex

7） 信号量对象
a) CreateSemaphore
b) WaitForSingleObject(hSem, INFINITE)
c) ReleaseSemaphore

参考流程框架

```
#define   N 100       /*number of slots in the buffer*/
typedef int semaphore; /*信号量*/
semaphore mutex=1;
semaphore empty=N;
semaphore full=0;

void producer(void){
    int item;
    while(TRUE){
        produce_item(&item);
        down(&empty);
        down(&mutex);
        enter_item(item);
        up(&mutex);
        up(&full);
    }
}

void consumer(void){
    int item;
```

```
    while(TRUE){
        down(&full);
        down(&mutex);
         remove_item(&item);
        up(&mutex);
        up(&empty);
        consume_item(item);
    }
}
```

参考实例

a) ipc_pc_v1.zip
b) ipc_pc_event_v1.zip

测试案例

a) 2 个生产者；3 个消费者；15 个产品；缓冲区大小为 4
b) 0 个生产者；3 个消费者；15 个产品；缓冲区大小为 4
c) 2 个生产者；0 个消费者；15 个产品；缓冲区大小为 4
d) 2 个生产者；3 个消费者；15 个产品；缓冲区大小为 1
e) 0 个生产者；0 个消费者；15 个产品；缓冲区大小为 4
f) 2 个生产者；3 个消费者；0 个产品；缓冲区大小为 4