東南大學
SOUTHEAST UNIVERSITY

操作系统实验报告

姓名：任杰文

学号：09013430

东南大学计算机科学与工程学院

School of Computer Science & Engineering

Southeast University

二0一六年四月二十六日

实验三

一、 实验内容：

在 Windows 和 Linux 操作系统上，利用各自操作系统提供的 Mutex 和信号量机制（Win32 API 或 Pthreads），实现生产者/消费者问题。具体要求见”Operating System Concepts(Seventh Edition)” Chapter 6 后的 Project（P236-241）。

二、 实验目的：

通过实验，理解 Win32 API、Pthreads 中 mutex locks、semaphores 等使用方法，并掌握如何利用它们实现进程（线程）间的同步和互斥。

具体要求：

1. 程序应独立完成编写和调试，严禁抄袭和拷贝！
2. 在完成实验内容后撰写实验报告，实验报告格式见《操作系统实验报告规范》。
3. **5 月 8 日晚 12 点之前**，将程序及实验报告提交助教。

三、 设计思路及流程图

1. 设计思路：

（1） 初始化信号量(full, empty, mutex)，缓冲区等。

（2） 创建生产者、消费者线程各 5 个，主线程睡眠 5000ms；

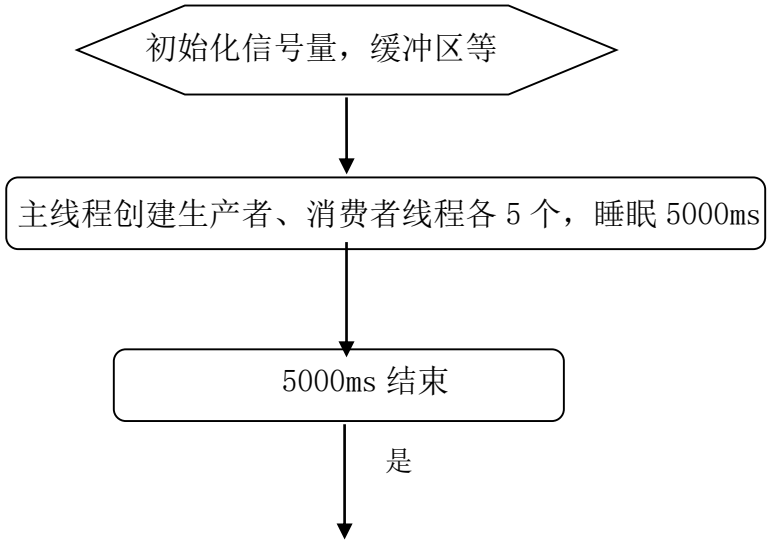
（3） 生产者生成 100 以下的随机数 x，睡眠 x 毫秒，并尝试将产生数插入缓冲区；

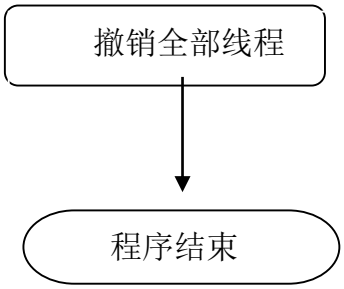
（4） 消费者生成 100 以下的随机数 x，睡眠 x 毫秒，并尝试从缓冲区读取数据；

（5） 5000 毫秒结束，主线程撤销所有生产者消费者线程。

（6） 程序退出；

2. 流程图：





四、 源程序

1. Windows 源程序：

```
#include <stdio.h>
#include <windows.h>
#include <iostream>
#include <time.h>
#include <vector>
using namespace std;

//定义缓冲区数据类型
typedef int buffer_item;
//定义缓冲区大小
#define BUFFER_SIZE 5
//创建缓冲区
buffer_item myBuffer[BUFFER_SIZE];
//定义信号量
HANDLE empty;
HANDLE full;
HANDLE mutex;
//定义生产者、消费之缓冲区下标
int in = 0, out = 0;

//初始化函数
void init()
{
	//初始化信号量
	empty = CreateSemaphore( NULL, 5, 5, NULL );
	full = CreateSemaphore( NULL, 0, 5, NULL );
	mutex = CreateSemaphore( NULL, 1, 1, NULL );
}

//插入一个数据到缓冲区
bool insert(buffer_item item)
{
	//获得锁
```

```
        WaitForSingleObject( empty, INFINITE );
        WaitForSingleObject( mutex, INFINITE );
        //插入数据
        myBuffer[ in ] = item;
        in = (in + 1) % 5;
        //释放锁
        ReleaseSemaphore( mutex, 1, NULL );
        ReleaseSemaphore( full, 1, NULL );
        return true;
}
//从缓冲区移除一个数据
bool remove(buffer_item *item)
{
        //获得锁
        WaitForSingleObject( full, INFINITE );
        WaitForSingleObject( mutex, INFINITE );
        //获得数据
        *item = myBuffer[out];
        out = (out + 1) % 5;
        //释放锁
        ReleaseSemaphore( mutex, 1, NULL );
        ReleaseSemaphore( empty, 1, NULL );
        return true;
}

//生产者
DWORD WINAPI producer(LPVOID   lpParam)
{
        int num = *(int*)lpParam;
        srand((unsigned)time(0));
        while (true)
        {
                //睡眠随机时间
                _sleep( rand() % 100 );
                //插入随机数
                int data = rand() % 100;
                //打印信息
                //第num个生产者
                printf("Producer%d produced %d\n", num, data);
                if ( insert(data) )
                        ;
                else
                        printf("Warning:produce error\n");
        }
```

```
}

//消费者
DWORD WINAPI consumer(LPVOID    lpParam)
{
        int num = *(int*)lpParam;
        while (true)
        {
                //睡眠随机时间
                _sleep( rand() % 100 );
                //插入随机数
                int data = 0;
                //打印信息
                if ( remove(&data) )
                        //第num个消费者
                        printf("Consumer%d consumed %d\n", num, data);
                else
                        printf("Warning:consumer error\n");
        }
}

int main(void)
{
        freopen("windowOut.txt","w",stdout);
        //初始化信号量
        init();
        //创建5 个生产者和个消费者线程
        HANDLE producerThread[5] , consumerThread[5];
        int num[5] = {0 ,1, 2, 3, 4};
        for( int i = 0; i < 5; i++)
        {
                producerThread[i] = CreateThread(NULL,0,producer,&num[i],0,NULL);
                consumerThread[i] = CreateThread(NULL,0,consumer,&num[i],0,NULL);
        }
        //睡眠一段时间
        _sleep(5000);
        //退出线程
        for( int i = 0; i < 1; i++)
        {
                CloseHandle( producerThread[i] );
                CloseHandle( consumerThread[i] );
        }
        return 0;
}
```

2. linux 源程序：

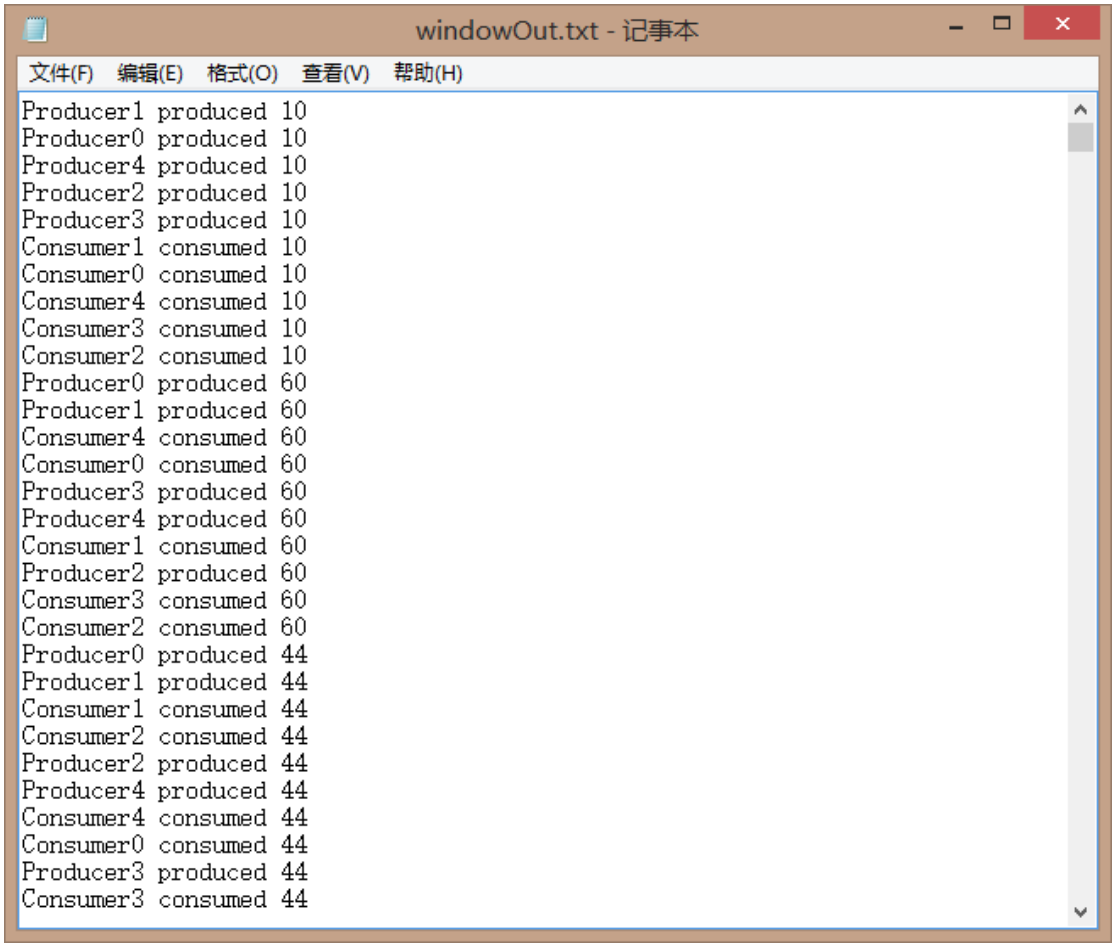
```
#include <stdio.h>
#include <unistd.h>
#include <semaphore.h>
#include <iostream>
#include <cstdlib>
#include <time.h>
#include <pthread.h>
using namespace std;

//定义缓冲区数据类型
typedef int buffer_item;
//定义缓冲区大小
#define BUFFER_SIZE 5
//创建缓冲区
buffer_item myBuffer[BUFFER_SIZE];
//定义信号量
pthread_mutex_t mutex;
sem_t empty;
sem_t full;
//定义生产者、消费之缓冲区下标
int in = 0, out = 0;

//初始化函数
void init()
{
	//初始化信号量
	pthread_mutex_init(&mutex,NULL);
	sem_init(&empty,0,5);
	sem_init(&full,0,0);
}

//插入一个数据到缓冲区
bool insert(buffer_item item)
{
	//获得锁
	sem_wait(&empty);
	pthread_mutex_lock(&mutex);
	//插入数据
	myBuffer[ in ] = item;
	in = (in + 1) % 5;
	//释放锁
	pthread_mutex_unlock(&mutex);
```

```
        sem_post(&empty);
        return true;
}
//从缓冲区移除一个数据
bool remove(buffer_item *item)
{
        //获得锁
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        //取数据
        *item = myBuffer[out];
        out = (out + 1) % 5;
        //释放锁
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        return true;
}
//生产者
void *producer(void *data)
{
        int num = *(int*)data;
        srand((unsigned)time(0));
        while (true)
        {
                //睡眠随机时间
                usleep( rand() % 100 );
                //插入随机数
                int data = rand() % 100;
                //打印信息
                if ( insert(data) )
                        //第num个生产者
                        printf("Producer%d produced %d\n", num, data);
                else
                        printf("Warning:produce error\n");
        }
}
//消费者
void *consumer(void *data)
{
        int num = *(int*)data;
        while (true)
        {
                //睡眠随机时间
                usleep( rand() % 100 );
```

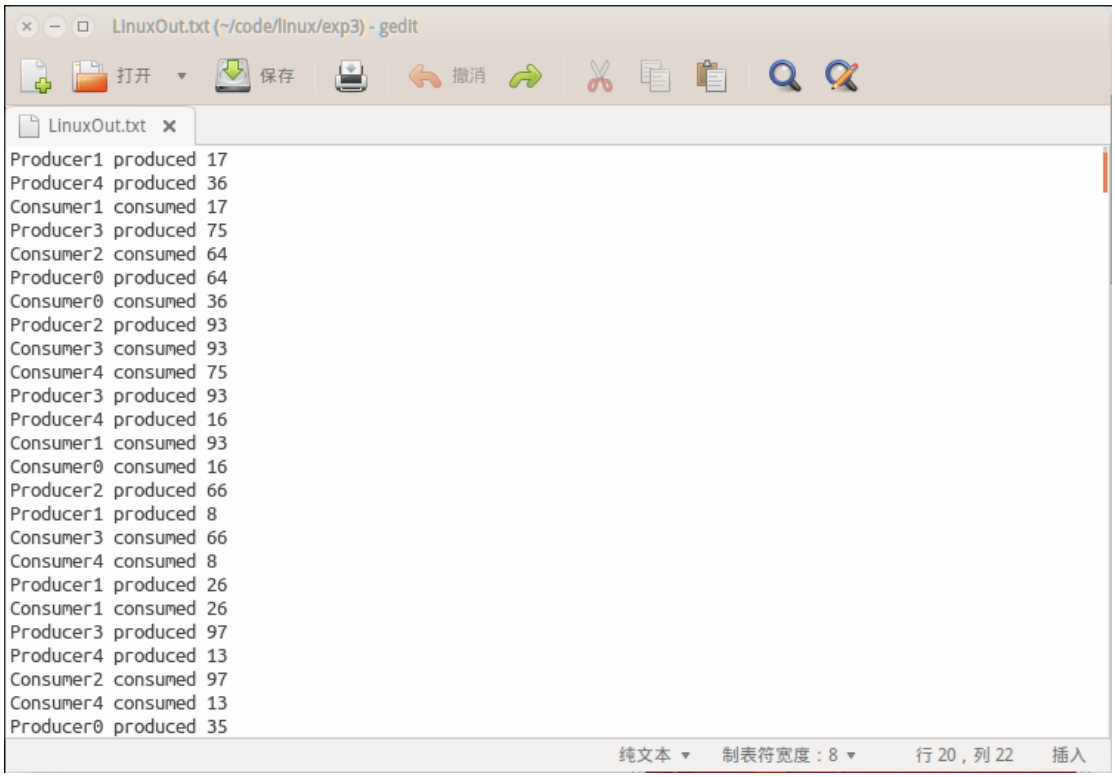
```
            //插入随机数
            int data = 0;
            //打印信息
            if ( remove(&data) )
                //第num个消费者
                printf("Consumer%d consumed %d\n", num, data);
            else
                printf("Warning:consumer error\n");
        }
    }

    int main(void)
    {
        freopen("LinuxOut.txt","w",stdout);
        //初始化信号量
        init();
        //创建 5 个生产者和5个消费者线程
        pthread_t producerThread[5] , consumerThread[5];
        pthread_attr_t producerAttr[5], consumerAttr[5];
        int num[5] = {0 ,1, 2, 3, 4};
        for( int i = 0; i < 5; i++)
        {
            pthread_attr_init(&producerAttr[i]);
            pthread_attr_init(&consumerAttr[i]);
            pthread_create(&producerThread[i], &producerAttr[0], producer, &num[i]);
            pthread_create(&consumerThread[i], &consumerAttr[0], consumer, &num[i]);
        }
        //睡眠一段时间
        usleep(5000);
        //退出线程
        for( int i = 0; i < 1; i++)
        {
            pthread_cancel(producerThread[i]);
            pthread_cancel(consumerThread[i]);
        }
        return 0;
    }
```

五、 实验截图

Windows：重定向提示输出：



Linux： 重定向提示输出：



六、 实验体会

1. 由于在 linux 下编程时，判断 producer 是否产生数据的语句在实际数据产生之后，而且此判断语句不在 cirtical_section 之内，所以有时 consumer 的提示语句反而比 producer 的提示语句更早打印。

   而在 windows 下编程时，判断 producer 是否产生数据的语句在实际数据产生之前，而且此判断语句不在 cirtical_section 之内，因此有时 producer 可能连续打印超出缓冲区大小的 producer 语句，原因是在虽然生产者还没有真正将生产的数据放入缓冲区，但其提示语句已经打印。

   这个问题的发现也让我对进程并发执行有了更直观和深刻的认识。因此提示信息的打印顺序实际上并不完全代表真正的生产消费顺序，只作为参考信息输出。

   解决方案：将打印语句放入 critical_section 即可。但由于课本给的参考代码是这个结构，所以还是按照课本来写了。

2. 通过这次实验，基本掌握了在 windows 下和 linux 下的多线程编程中的同步互斥的使用方法，熟练了线程同步中信号量和互斥锁在实际应用中的使用。对于多线程有了更为深刻的理解和认识。

3. 实验背景

生产者-消费者问题是一个经典的进程同步问题，该问题最早由 Dijkstra 提出，用以演示他提出的信号量机制。在同一个进程地址空间内执行的两个线程生产者线程生产物品，然后将物品放置在一个空缓冲区中供消费者线程消费。消费者线程从缓冲区中获得物品，然后释放缓冲区。当生产者线程生产物品时，如果没有空缓冲区可用，那么生产者线程必须等待消费者线程释放出一个空缓冲区。当消费者线程消费物品时，如果没有满的缓冲区，那么消费者线程将被阻塞，直到新的物品被生产出来。

4. 实验所用函数（linux）

（1） pthread_create

```
extern int pthread_create __P ((pthread_t *__thread, __const
pthread_attr_t *__attr,
    void *(*__start_routine) (void *),  void *__arg));
```

第一个参数为指向线程标识符的指针，第二个参数用来设置线程属性，第三个参数是线程运行函数的起始地址，最后一个参数是运行函数的参数。这里，我们的函数 thread 不需要参数，所以最后一个参数设为空指针。第二个参数我们也设为空指针，这样将生成默认属性的线程。对线程属性的设定和修改我们将在下一节阐述。当创建线程成功时，函数返回 0,若不为 0 则说明创建线程失败，常见的错误返回代码为 EAGAIN 和 EINVAL.前者表示系统限制创建新的线程，例如线程数目过多了;后者表示第二个参数代表的线程属性值非法。创建线程成功后，新创建的线程则运行参数三和参数四确定的函数，原来的线程则继续运行下一行代码。

（2） pthread_mutex_init

函数原型：int pthread_mutex_init(pthread_mutex_t *restrict mutex,const pthread_mutexattr_t *restrict attr);

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_init()函数是以动态方式创建互斥锁的，参数 attr 指定了新建互斥锁的属性。如果参数 attr 为空，则使用默认的互斥锁属性，默认属性为快速互斥锁 。互斥锁的属性在创建锁的时候指定，在 LinuxThreads 实现中仅有一个锁类型属性，不同的锁类型在试图对一个已经被锁定的互斥锁加锁时表现不同。

pthread_mutexattr_init()函数成功完成之后会返回零，其他任何返回值都表示出现了错误。

函数成功执行后，互斥锁被初始化为未锁住态。

（3） pthread_attr_init

int pthread_attr_init(pthread_attr_t *attr);

pthread_attr_init，函数，作用是初始化一个线程对象的属性。

（4） pthread_cancel

int pthread_cancel(pthread_t thread)

检查本线程是否处于 Canceld 状态，如果是，则进行取消动作，否则直接返回。 此函数在线程内执行，执行的位置就是线程退出的位置，在执行此函数以前，线程内部的相关资源申请一定要释放掉，他很容易造成内存泄露。

5. 实验所用函数（windows）

（1） CreateThread

Windows API 函数。该函数在主线程的基础上创建一个新线程。微软在 Windows API 中提供了建立新的线程的函数 CreateThread。

HANDLE CreateThread(

LPSECURITY_ATTRIBUTES lpThreadAttributes,

DWORD dwStackSize,

LPTHREAD_START_ROUTINE lpStartAddress,

LPVOID lpParameter,

DWORD dwCreationFlags,

LPDWORD lpThreadId);

参数说明：

lpThreadAttributes：指向 SECURITY_ATTRIBUTES 型态的结构的指针。在 Windows 98 中忽略该参数。在 Windows NT 中，它被设为 NULL，表示使用缺省值。

dwStackSize，线程堆栈大小，一般=0，在任何情况下，Windows 根据需要动态延长堆栈的大小。

lpStartAddress，指向线程函数的指针，形式：@函数名，函数名称没有限制，但是必须以下列形式声明：

DWORD WINAPI ThreadProc (LPVOID pParam) ，格式不正确将无法调用成功。

lpParameter：向线程函数传递的参数，是一个指向结构的指针，不需传递参数时，为 NULL。

dwCreationFlags ：线程标志,可取值如下

CREATE_SUSPENDED： 创建一个挂起的线程

0 ：创建后立即激活。

lpThreadId:保存新线程的 id。

（2） WaitForSingleObject

```
    DWORD WaitForSingleObject(
HANDLE hObject, //指明一个内核对象的句柄
DWORD dwMilliseconds); //等待时间
```

该函数需要传递一个内核对象句柄，该句柄标识一个内核对象，如果该内核对象处于未通知状态，则该函数导致线程进入阻塞状态；如果该内核对象处于已通知状态，则该函数立即返回 WAIT_OBJECT_0。第二个参数指明了需要等待的时间（毫秒），可以传递 INFINITE 指明要无限期等待下去，如果第二个参数为 0，那么函数就测试同步对象的状态并立即返回。如果等待超时，该函数返回 WAIT_TIMEOUT。如果该函数失败，返回 WAIT_FAILED。

（3） ReleaseSemaphore

ReleaseSemaphore

```
ReleaseSemaphore(
    __in      HANDLE hSemaphore,
    __in      LONG lReleaseCount,
    __out_opt LPLONG lpPreviousCount
    );
```

lReleaseCount 增加的信号值。