电子科技大学

计算机专业类课程

实验报告

课程名称：操作系统

学　　院：计算机科学与工程

专　　业：计算机科学与技术

学生姓名：韩会彬

学　　号：2013060105004

指导教师：薛瑞尼

日　　期：2016 年 06 月　05 日

电 子 科 技 大 学

实 验 报 告

实验一

一、实验名称：生产者消费者问题

二、实验学时：2

三、实验内容和目的：

共享缓冲区中放置一个数字，取值范围为[0, 10]，初值为 0。生产者将此值加 1，消费者将此值减 1。

1. 场景 1
    - 同一进程内启动一组生产者线程和一组消费者线程
    - 缓冲区为本进程的全局变量

2. 场景 2
    - 启动一组生产者进程和一组消费者进程
    - 同一个数据文件为缓冲区

- 输入
    - p：生产者数量
    - c：消费者数量

- 输出

打印当前共享缓冲区中的数值，或者生产者消费者的状态。

四、实验原理：

生产者消费者模型：

- 生产者：满则等待，空则填充
- 消费者：空则等待，有则获取
- 不允许同时进入缓冲区
- 多个生产者，多个消费者
- 一个共享缓冲区

生产者/消费者问题的启示

- 资源数量：资源信号量
- 资源访问：互斥信号量
- 先申请资源，再申请访问权
- 资源信号量 P、V 操作分布在不同进程
- 互斥信号量 P、V 操作出现在同一进程

五、实验步骤和结果：

多线程实验代码：

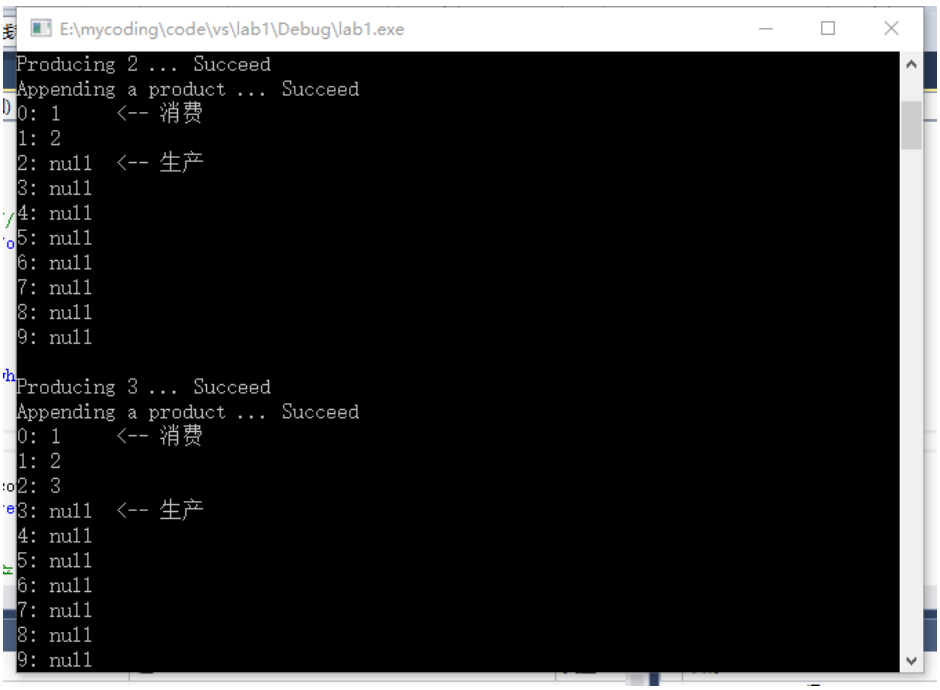
```
#include <windows.h>
#include <iostream>
#include<stdio.h>
#include<stdlib.h>

using namespace std;

const unsigned short SIZE_OF_BUFFER = 10;    //缓冲区长度
unsigned short ProductID = 0;                //产品号
unsigned short ConsumeID = 0;                //将被消耗的产品号
unsigned short in = 0;                       //产品进缓冲区时的下标
unsigned short out = 0;                      //产品出缓冲区时的下标

int g_buffer[SIZE_OF_BUFFER];                //缓冲区是个循环队列
bool g_continue = true;                      //控制程序结束
HANDLE g_hMutex;                             //用于线程间的互斥
HANDLE g_hFullItems;                         //缓冲区中被占用的项
HANDLE g_hEmptyItems;                        //缓冲区中的空项

DWORD  WINAPI Producer(LPVOID);               //生产者线程
DWORD  WINAPI Consumer(LPVOID);               //消费者线程

int main()
{
    //创建各个互斥信号
    g_hMutex = CreateMutex(NULL, FALSE, NULL);
    g_hFullItems = CreateSemaphore(NULL, 0, SIZE_OF_BUFFER, NULL);//创建信号
    g_hEmptyItems = CreateSemaphore(NULL, SIZE_OF_BUFFER,
        SIZE_OF_BUFFER, NULL);
    //缓冲区初始化
    for (int i = 0; i< SIZE_OF_BUFFER; ++i){
        g_buffer[i] = -1;    //当值为-1时该项为空
    }
```

```
    const unsigned short PRODUCERS_COUNT=3;        //生产者的个数
    const unsigned short CONSUMERS_COUNT=5;        //消费者的个数

    //总的线程数
    const unsigned short THREADS_COUNT = PRODUCERS_COUNT + CONSUMERS_COUNT;
    HANDLE hThreads[THREADS_COUNT];      //各线程的handle
    DWORD producerID[PRODUCERS_COUNT];  //生产者线程的标识符
    DWORD consumerID[CONSUMERS_COUNT];  //消费者线程的标识符

    //创建生产者线程
    for (int i = 0; i<PRODUCERS_COUNT; ++i){
        hThreads[i]
            = CreateThread(NULL, 0, Producer, NULL, 0, &producerID[i]);//window提供的
API函数
        if (hThreads[i] == NULL) return -1;                          //创建一个线程
    }
    //创建消费者线程
    for (int i = 0; i<CONSUMERS_COUNT; ++i){
        hThreads[PRODUCERS_COUNT + i]
            = CreateThread(NULL, 0, Consumer, NULL, 0, &consumerID[i]);
        if (hThreads[i] == NULL) return -1;
    }

    while (g_continue){
        if (getchar()){  //按回车后终止程序运行
            g_continue = false;
        }
    }
    cout << "over";
    return 0;
}

//生产一个产品
void Produce()
{
    cout << endl << "Producing " << ++ProductID << " ";
    cout << "Succeed" << endl;
}

//把新生产的产品放入缓冲区
void Append()
{
    cout << "Appending a product ... ";
    g_buffer[in] = ProductID;
```

```
    in = (in + 1) % SIZE_OF_BUFFER;
    cout << "Succeed" << endl;
    //输出缓冲区当前的状态
    for (int i = 0; i<SIZE_OF_BUFFER; ++i){
        cout << i << ": ";
        if (g_buffer[i] == -1)
            cout << "null";
        else
            cout << g_buffer[i];
        if (i == in) cout << '\t' << " <-- 生产";
        if (i == out) cout << '\t' << " <-- 消费";
        cout << endl;
    }
}

//从缓冲区中取出一个产品
void Take()
{
    cout << endl << "Taking a product ... ";
    ConsumeID = g_buffer[out];
    g_buffer[out] = -1;
    out = (out + 1) % SIZE_OF_BUFFER;
    cout << ConsumeID << "--Succeed" << endl;

    //输出缓冲区当前的状态
    for (int i = 0; i<SIZE_OF_BUFFER; ++i){
        cout << i << ": ";
        if (g_buffer[i] == -1)
            cout << "null";
        else
            cout << g_buffer[i];
        if (i == in) cout << '\t' << " <-- 生产";
        if (i == out) cout << '\t' << " <-- 消费";
        cout << endl;
    }
}

//消耗一个产品
void Consume()
{
    cout << "Consuming " << ConsumeID << " ... ";
    cout << "Succeed" << endl;
}
```

```
//生产者
DWORD  WINAPI Producer(LPVOID lpPara)
{
    while (g_continue){
        int i = rand() % 5;
        Sleep(i * 1000);
        WaitForSingleObject(g_hEmptyItems, INFINITE);//等待信号灯
        WaitForSingleObject(g_hMutex, INFINITE);
        Produce();
        Append();
        ReleaseMutex(g_hMutex);
        ReleaseSemaphore(g_hFullItems, 1, NULL);
    }
    return 0;
}

//消费者
DWORD  WINAPI Consumer(LPVOID lpPara)
{
    while (g_continue){
        int i = rand() % 5;
        Sleep(i * 1000);
        WaitForSingleObject(g_hFullItems, INFINITE);
        WaitForSingleObject(g_hMutex, INFINITE);
        Take();
        Consume();
        ReleaseMutex(g_hMutex);
        ReleaseSemaphore(g_hEmptyItems, 1, NULL);
    }

    return 0;
}
```

运行结果：



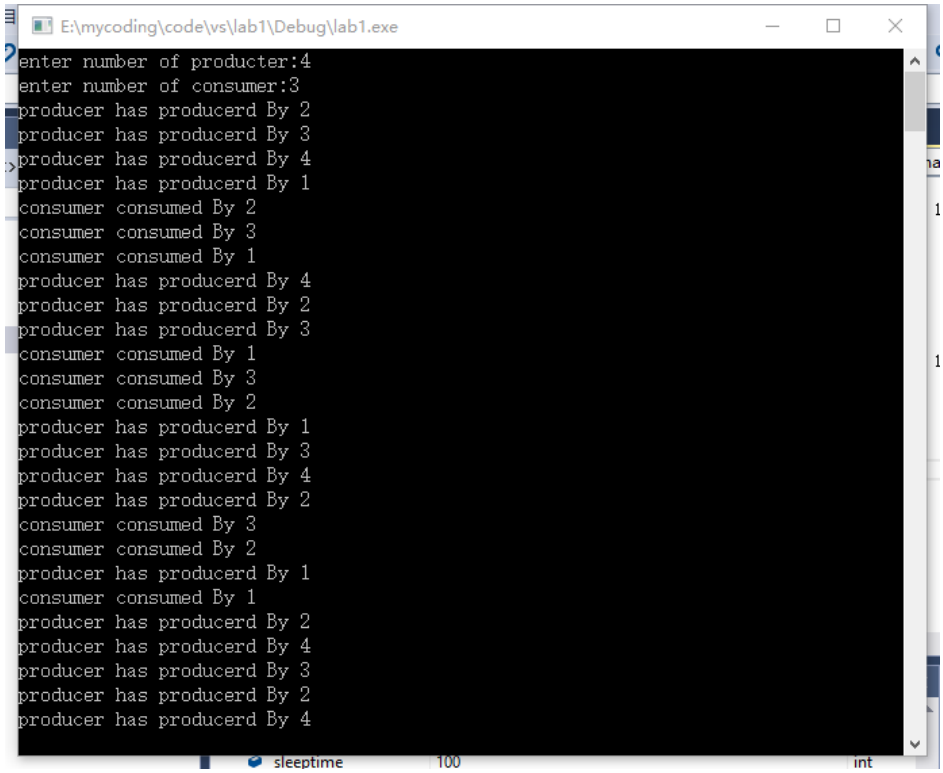
多进程实验代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<windows.h>
#define BUFFER_SIZE 10
typedef int buffer_item;
struct v
{
    int i;
};

buffer_item buffer[BUFFER_SIZE + 1];
buffer_item front = 0, rear = 0;
HANDLE mutex, empty, full;

DWORD WINAPI producer(PVOID Param)
{
    struct v data = *(struct v *)Param;
    srand((unsigned)time(0));
    while (1)
    {
        Sleep(rand() % 11);
        WaitForSingleObject(empty, INFINITE);
        WaitForSingleObject(mutex, INFINITE);
```

```
            printf("producer has producerd By %d\n", data.i);
            ReleaseMutex(mutex);
            ReleaseSemaphore(full, 1, NULL);
        }
}

DWORD WINAPI consumer(PVOID Param)
{
    struct v data = *(struct v *)Param;
    srand((unsigned)time(0));
    while (1)
    {
        Sleep(rand() % 11);
        WaitForSingleObject(full, INFINITE);
        WaitForSingleObject(mutex, INFINITE);
        printf("consumer consumed By %d \n", data.i);
        ReleaseMutex(mutex);
        ReleaseSemaphore(empty, 1, NULL);
    }
}

int main(int argc, char *argv[])
{

    int sleeptime, pnum, snum;
    DWORD *ThreadIdP, *ThreadIdS, i;
    struct v *countp, *counts;
    HANDLE *ThreadHandleP, *ThreadHandleS;

    sleeptime = 100;
    printf("enter number of producter:");
    scanf_s("%d",&pnum);
    printf("enter number of consumer:");
    scanf_s("%d", &snum);
    ThreadHandleP = (HANDLE *)malloc(pnum * sizeof(HANDLE));
    ThreadHandleS = (HANDLE *)malloc(snum * sizeof(HANDLE));
    ThreadIdP = (DWORD *)malloc(pnum * sizeof(DWORD));
    ThreadIdS = (DWORD *)malloc(pnum * sizeof(DWORD));
    mutex = CreateMutex(NULL, FALSE, NULL);
    empty = CreateSemaphore(NULL, BUFFER_SIZE, BUFFER_SIZE, NULL);
    full = CreateSemaphore(NULL, 0, BUFFER_SIZE + 1, NULL);
    /*生产者*/
    countp = (struct v *)malloc((pnum + 1)*sizeof(struct v));
    for (i = 0; i<pnum; i++)
```

```
    {
        countp[i + 1].i = i + 1;
        ThreadHandleP[i] = CreateThread(NULL, 0, producer, &countp[i + 1], 0,
&ThreadIdP[i]);
    }
    /*消费者*/
    counts = (struct v *)malloc((snum + 1)*sizeof(struct v));
    for (i = 0; i<snum; i++)
    {
        counts[i + 1].i = i + 1;
        ThreadHandleS[i] = CreateThread(NULL, 0, consumer, &counts[i + 1], 0,
&ThreadIdS[i]);
    }
    /*运行时间*/
    Sleep(sleeptime);

    printf("over");
    return 0;
}
```

运行结果：



六、实验结论和心得：

通过本次实验，进一步深入了解了生产者与消费者问题，由于以前没有接触过进程 API 方面的知识，这次实验中参考网上的文章和程序才得以完成，使我了解了自己编程能力的不足，经过实验提升了自己的能力。