

**计算机与信息 学院实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验课程： | 操作系统实验 | | | | |
| 实验编号： | 实验三 | | | | |
| 实验名称： | 生产者/消费者问题 | | | | |
| 实验人员： | 学号 | 18111207248 | | | |
| 姓名 | 吴钰 | | | |
| 班级 | 18计算机科学与技术创新班 | | | |
| 实验日期： | 2020.5.10 | | | | |
| 实验室： |  | | | | |
|  |  | | | | |
| 实验评价： |  | | | | |
| 实验成绩： | |  | 评价日期： |  |
|  | 指导教师： | |  | | |

# 一、实验目的

## 1. 通过实验，掌握Windows环境下互斥锁和信号量的实现方法，进一步加深对进程同步机制的理解

## 二、实验要求

# 1．利用线程模拟进程；

# 2. 可视化显示模拟的同步。

# 3. 实验报告中需分析所得结果，指出某消费者线程或生产者线程在何时被阻塞、何时被唤醒 。

# 三、设计

【定义所有抽象数据类型，自定义函数的功能详细描述（伪代码表示），以及主程序的流程图。】

生产者线程

生产者线程

#include <stdio.h>

#include <process.h>//\_beginthread函数

#include <windows.h>

#include <time.h>

//srand(time(NULL));

//x=rand()%6;

using namespace std;

#define N 5

//可视化设置队列

int buff[N];

/\* 有N个槽数的缓冲区buf[N]，并实现循环缓冲队列 \*/

int front=0, rear=0;

/\*

ReleaseSemaphore(mutex,1,NULL);//v操作

WaitForSingleObject(mutex,INFINITE);//p操作

注意：在使用WaitForSingleObject等待信号量句柄时，

若信号量为signal状态，则wait过后信号量自动减1，

直到使用ReleaseSemaphore释放信号量，信号量才可增加

\*/

int x;

int p\_num=0;//初始生产产品数量

int c\_num=0;//初始消费产品数量

HANDLE mutex=CreateSemaphore(NULL,1,1,NULL);//Handle是资源的标识，操作系统要管理和操作这些资源，都是通过句柄来找到对应的资源

//信号量初值1，最大值1

HANDLE full=CreateSemaphore(NULL,0,5,NULL);//CreateSemaphore是创建信号量函数 头文件<windows.h>

HANDLE empty=CreateSemaphore(NULL,5,5,NULL);//第2、3参数：初始资源数量，最大并发数量

void produce\_item()

//生产产品

{

printf("生产第%d个产品 \n",++p\_num);

}

void enter\_item()

//放入缓存区

{

printf("放入缓存区第[%d]\n",rear);

buff[rear]=1;

rear=(rear+1)%N;

if(rear==front)

printf("缓存区已满\n");

}

void remove\_item()

//从缓存区中取

{

printf(" 从缓存区中取[%d] \n",front);

buff[front]=1;

front=(front+1)%N;

}

void consume\_item()

//消费产品

{

printf(" 消费产品第%d个商品\n",++c\_num);

}

/\* 生产者 \*/

void producer(PVOID param)

{

while(1){

Sleep(1000);//隔1000毫秒

produce\_item();

WaitForSingleObject(empty,INFINITE);//p操作 INFINITE：对象被触发信号后，函数才会返回

WaitForSingleObject(mutex,INFINITE);//p操作

enter\_item(); /\* 将一个新的数据项放入缓冲区 \*/

ReleaseSemaphore(mutex,1,NULL);//v操作

ReleaseSemaphore(full,1,NULL);//v操作

}

}

/\* 消费者 \*/

void consumer(PVOID param)

{

while(1){

x=rand()%6+1;//1<=x<=6

printf("\n%d\n",x);

Sleep(x\*1000);

WaitForSingleObject(full,INFINITE);//p操作

WaitForSingleObject(mutex,INFINITE);//p操作

remove\_item(); /\* 从缓冲区中取走一个数据项 \*/

ReleaseSemaphore(mutex,1,NULL);//v操作

ReleaseSemaphore(empty,1,NULL);//v操作

consume\_item(); /\* 对数据项进行操作（消费）\*/

}/\* 对数据项进行操作（消费）\*/

}

int main(){

srand((unsigned)time(0));

\_beginthread(producer, 0, NULL);

\_beginthread(consumer, 0, NULL);

Sleep(15000);

printf("\nend\n") ;

return 0;

}

# 四、调试分析和思考

【(1) 调试过程中遇到的问题是如何解决的以及对设计与实现中关键点的回顾讨论和分析；(2) 算法的时空分析(包括基本操作和其他算法的时间复杂度和空间复杂度的分析)和改进设想；(3) 经验和体会等。】

在进行运行调试时我所注意到的问题：

1、在缓冲区为空时，消费者不能再进行消费；

2、在缓冲区为满时，生产者不能再进行生产；

3、在一个线程进行生产或消费时，其余线程不能再进行生产或消费等操作，即保持线程间的同步；

4、注意条件变量与互斥锁的顺序。

在访问共享区资源时，为避免多个线程同时访问资源造成混乱，需要对共享资源加锁，从而保证某一时刻只有一个线程在访问共享资源。

不能将线程里两个wait的顺序调换否则会出现死锁。如将consumer的两个wait调换，在producer发出signal信号后，如果producer线程此时再次获得运行机会，执行完了wait(space)，此时另一个consumer线程获得运行机会，执行了wait(mutex)如果此时缓冲区为空，那么consumer将会阻塞在wait(items)，而producer也会因为无法获得锁的所有权所以阻塞在wait(mutex)，这样两个线程都在阻塞，也就造成了死锁。

# 五、测试数据与结果

【列出你的测试结果，明确输入和输出数据。测试数据应该完整和严格，可以直接贴结果图。】

我在设计编程时，首先设定缓冲区大小为5

生产者和消费者相互等效,只要缓冲池未满,生产者便可将消息送入缓冲池;只要缓冲池未空,消费者便可从缓冲池中取走一个消息。

如果当前资源计数等于0，那么信号量处于未触发状态；那么系统会让调用线程进入等待状态。

CreateSemaphore(NULL,0,1,NULL); 当第二个参数为0时，调用线程就会进入等待状态。

当五个缓冲区被占满时，生产者线程会进入阻塞状态，直到有消费者从缓冲区中取出物品，生产者线程会被唤醒；当缓冲区为空时，消费者进程会被阻塞，直到有生产者生产物品并放入缓冲区后才会将消费者进程唤醒。



