学 号：	0121710880223	成 绩：	

武漢理工大學

课内实践报告

课程名称	操作系统
学　　院	计算机科学与技术学院
专　　业	软件工程专业
班　　级	软件 1702
姓　　名	刘佳迎
指导教师	刘 军

2019 ——2020 学年　第 1 学期

课内实践任务书

题目：实现生产者消费者（Bounded - Buffer Problem）问题

初始条件：学习了高级语言程序设计、汇编语言、数据结构、计算机组成原理课程，掌握了一种计算机高级语言。

要求完成的主要任务：（包括课程设计工作量及其技术要求，以及说明书撰写等具体要求）

通过研究 Linux 的线程机制和信号量实现生产者消费者（Bounded Buffer）问题的并发控制。

实验条件要求：每人一台与 Linux 主机联网的 Windows 主机,普通用户权限。

时间安排：

序号	阶段内容	所需时间
1	消化资料、系统设计	1 天
2	编程、调试	3 天
3	撰写报告	1 天
合计		5 天

指导教师签名：刘军　　年　月　日

系主任（或责任教师）签名：　　年　月　日

设计题目 实现生产者和消费者

一、 设计目的

1. 了解信号量的使用
2. 加深对信号量机制的理解
3. 通过研究 Linux 的进程机制和信号量实现生产者消费者问题的并发控制
4. 掌握基本的同步互斥算法，理解生产者与消费者模型
5. 了解多线程（多进程）的并发执行机制，线程（进程）间的同步与互斥

二、 设计要求

1. 了理解生产者与消费者问题模型，掌握解决问题的算法思想
2. 掌握正确使用同步机制的方法
3. 个生产者和消费者对有界缓冲区进行操作后, 即时显示有界缓冲区的全部内容, 当前指针位置和生产者/消费者线程的标识符.
4. 生产者和消费者各有两个以上.
5. 多个生产者或多个消费者之间须有共享对缓冲区进行操作的函数代码.。

三、 设计思想及原理

3.1 问题描述

生产者消费者问题（Producer-consumer problem），也称有限缓冲问题（Bounded-buffer problem），是一个多线程同步问题的经典案例。该问题描述了两个共享固定大小缓冲区的线程——即所谓的“生产者”和“消费者”——在实际运行时会发生的问题。生产者的主要作用是生成一定量的数据放到缓冲区中，然后重复此过程。与此同时，消费者也在缓冲区消耗这些数据。该问题的关键就是要保证生产者不会在缓冲区满时加入数据，消费者也不会在缓冲区中空时消耗数据。

3.2 解决方案

要解决该问题，就必须让生产者在缓冲区满时休眠（要么干脆就放弃数据），等到下次消费者消耗缓冲区中的数据的时候，生产者才能被唤醒，开始往缓冲区添加数据。同样，也可以让消费者在缓冲区空时进入休眠，等到生产者往缓冲区添加数据之后，再唤醒消费者。通常采用进程间通信的方法解决该问题，常用的方法有信号灯法等。如果

解决方法不够完善，则容易出现死锁的情况。出现死锁时，两个线程都会陷入休眠，等待对方唤醒自己。该问题也能被推广到多个生产者和消费者的情形。

3.3 知识点

3.3.1 进程的同步与互斥

互斥: 一组并发进程中的一个或多个程序段，因共享某一公有资源而导致它们必须以一个不允许交叉执行的单位执行。也就是说，不允许两个以上的共享该资源的并发进程同时进入临界区。

同步: 把异步环境下的一组并发进程因直接制约而互相发送消息而进行互相合作、互相等待，使得各进程按一定的速度执行的过程称为进程间的同步。

3.3.2 信号量和 P \V 操作

信号量（semaphore）的数据结构为一个值和一个指针，指针指向等待该信号量的下一个进程。信号量的值与相应资源的使用情况有关。当它的值大于 0 时，表示当前可用资源的数量；当它的值小于 0 时，其绝对值表示等待使用该资源的进程个数。注意，信号量的值仅能由 PV 操作来改变。其中 S 表示信号量的值，P 表示 P 操作，V 表示 V 操作；

P（S）：

1. 将信号量 S 的值减 1，即进行 S = S-1；
2. 如果 S < 0，则该进程进入阻塞队列；
3. 如果 S >= 0, 则该进程继续执行；
4. 执行一次 P 操作其实就是意味请求分配一个资源，所以针对 □ 和 □ 来说就好理解了，当信号量的值小于 0，那么就表示没有可用资源，那么进程就只能进行等待其他拥有该资源的进程释放资源之后，才能进行执行；当信号量大于 0 的时候，那么表示还有足够的资源，所以，当前进程就可以继续执行；

V（S）：

1. 将信号量 S 的值加 1，即 S = S + 1；
2. 如果 S > 0，则该进程继续执行；
3. 如果 S < 0, 则释放阻塞队列中的第一个等待信号量的进程；
4. 执行一次 V 操作其实就是意味释放一个资源，所以针对 □ 和 □ 来说就好理解了，当信号量的值大于 0，那么就表示有可用资源，那么表示信号量的资源足够进程进行申请，就不需要将进程进行放入到阻塞队列中；而当信号量小于 0 的时候，就表示针对这个信号量，还有其他的进程是已经进行了申请信号量的操作，而只是之前是无法满足进程获取资源的，简单点说，就是表示阻塞队列中还有其他的进程是

执行了 P 操作，在等待信号量，所以，这样的话，就讲阻塞队列中的第一个等待信号量的进程进行处理即可；

3.3.3 实验环境配置

本次实验采用 Ubuntu 18.0.4 版本虚拟机, 并使用 c++ 语言来完成。由于系统并没有安装多线程库 (pthread), 因此需要键入以下命令搭建环境。

```
sudo apt-get install glibc-doc
sudo apt-get install manpages-posix-dev
```

四、 数据结构以及模块说明

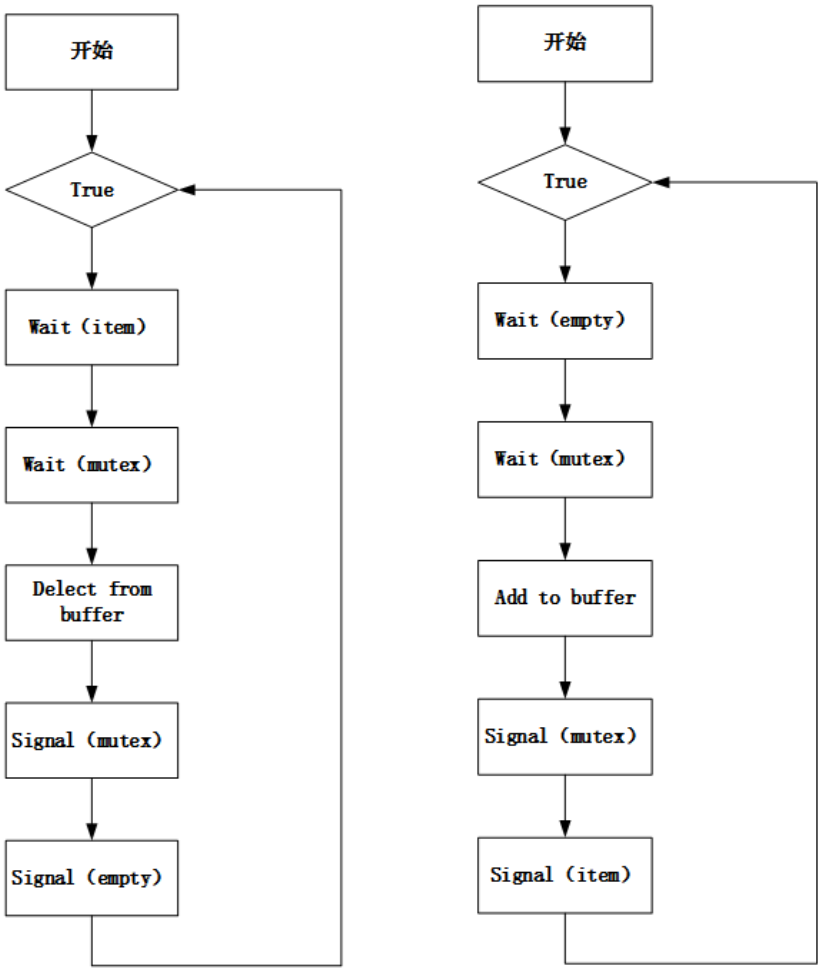

图 **1**　生产者、消费者流程图

假设缓冲区大小为 10，生产者、消费者线程若干。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。我们使用一个互斥量保护生产者向缓冲区中存入数据。由于有多个生产者，因此需要记住现在向缓冲区中存入的位置。使用一个互斥量保护缓冲区中消息的数目，这个生产的数据数目作为生产者和消费者沟通的桥梁。

使用一个条件变量用于唤醒消费者。由于有多个消费者，同样消费者也需要记住每次取的位置。

生产者代码块

```
void *producer_function(void *args)
{
    int index = *(int*)args;
    while(1){
        pthread_mutex_lock(&mutex);
        while(g_count >= MAXCAPACITY){
            printf("产品已满，生产者%d等待中\n", index);
            pthread_cond_wait(&condCon, &mutex);
        }

        ++g_count;
        printf("生产者%d生产了1个产品，现在共有%d个产品\n", index, g_count);
        pthread_cond_signal(&condPro);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}
```

消费者代码块

```
void *producer_function(void *args)
{
    int index = *(int*)args;
    while(1){
        pthread_mutex_lock(&mutex);
        while(g_count >= MAXCAPACITY){
            printf("产品已满，生产者%d等待中\n", index);
            pthread_cond_wait(&condCon, &mutex);
        }

        ++g_count;
        printf("生产者%d生产了1个产品，现在共有%d个产品\n", index, g_count);
        pthread_cond_signal(&condPro);
        pthread_mutex_unlock(&mutex);
    }
```

```
    pthread_exit(NULL);
}
```

在流程图中，两个 wait() 的顺序是不能调换的。如果调换语句顺序，在生产者发出 signal 信号后，如果生产者再次获得运行机会，执行完了 wait。与此同时一个消费者也执行了 wait，但是如果此时缓冲区为空，那么消费者将会发生阻塞，然后生产者也因为无法获得锁而导致阻塞，从而引起死锁。

在程序中，出现了流程不存在的一个额外的 while 循环。这是为了防止出现假死现象，即唤醒的是同类线程。假设当前多个生产者线程会调用 wait 方法阻塞等待，当其中的生产者线程获取到对象锁之后通知处于 WAITTING 状态的线程，如果唤醒的仍然是生产者线程，就会造成所有的生产者线程都处于等待状态。所以程序中添加一个 while 循环来防止假死的产生。

在程序中还可以看到，互斥锁和条件变量的顺序并不和流程图中的相同。在代码实现中，先添加的互斥锁，再进行 wait() 的循环。

原因如下，在流程图中的 wait()、signal() 操作实质上就是操作系统中的 PV 操作，根据 PV 操作的定义，该操作语句是原子操作。不会因多线程同时访问共享资源而造成混乱，所以在程序中为了保证线程之间的同步，需要先添加条件变量，然后才进行线程相应的操作，此时互斥锁的作用也是保证共享资源不会被其他线程访问。由于互斥锁的存在，顺序的改变也不会造成死锁的现象。

五、 运行结果

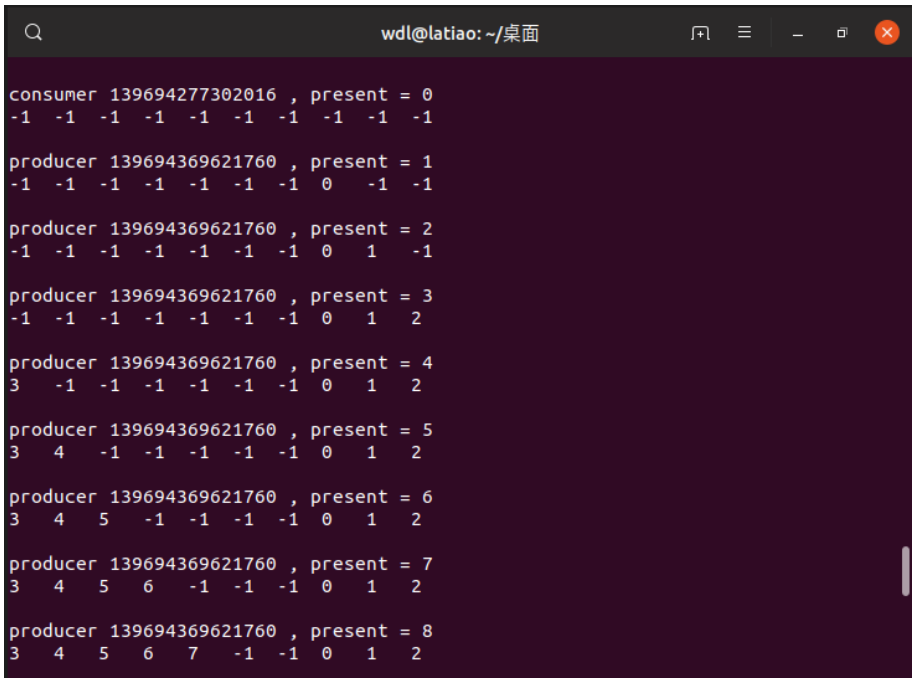

图 **2** 运行结果

六、 课程总结

本次实验学习了设计到在 linux 系统上多个 c++ 文件的编译，因此学习了如何使用 makefile 编写多个 c++ 文件。

这次的课程设计，主要深入理解了关于生产者消费者问题之间互斥和同步的问题，虽然在课堂上已经理解了生产者消费者问题的原理和解决思想，但是在这次的实践中才发现了许多课堂上没有思考到的问题，比如前面提到的假死，还有互斥锁如何使用。在这次课程设计中，我意识到了从理解的原理到实际代码的编写，需要考虑很多现实的问题，从原理到现实的转换是需要付出很多的。生产者消费者问题，其实就是一种有限缓存问题。在缓存区有限的情况下如何进行分配，如何防止死锁的发生。而且由于是在 Linux 系统上进行的操作，我对 Linux 系统的操作熟练度也越发提高，也渐渐了解到了 Linux 系统受编程者欢迎的原因。总的来说这次课程设计收获很多，我的编程水平得到了提高。

附录 A 设计代码

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

```c
#include <pthread.h>
#include <semaphore.h>
#include <string.h>

#define CONSUMER_NUM 3
#define PRODUCER_NUM 3
#define MAXCAPACITY 10

pthread_mutex_t mutex;
pthread_cond_t condCon;
pthread_cond_t condPro;
int g_count = 0;
int proIndex[PRODUCER_NUM];
int conIndex[CONSUMER_NUM];

void *producer_function(void *args);
void *consumer_function(void *args);

int main()
{
    pthread_t producer_thread[PRODUCER_NUM];
    pthread_t consumer_thread[CONSUMER_NUM];
    int res;
    int i;
    void *thread_result;

    pthread_mutex_init(&mutex,NULL);
    pthread_cond_init(&condCon,NULL);
    pthread_cond_init(&condPro,NULL);

    for(i = 0;i < PRODUCER_NUM; ++ i)
    {
        proIndex[i] = i + 1;
        res = pthread_create(&producer_thread[i], NULL, producer_function,
            (void*)&proIndex[i]);
        if(res != 0)
        {
            perror("create thread producer error");
            exit(EXIT_FAILURE);
        }
        printf("Starting producer %d success!\n", i);
    }

    for(i = 0; i < CONSUMER_NUM; ++ i)
    {
        conIndex[i] = i + 1;
        res = pthread_create(&consumer_thread[i], NULL, consumer_function,
```

```
            (void*)&conIndex[i]);
        if(res != 0)
        {
            perror("create thread producer error");
            exit(EXIT_FAILURE);
        }
        printf("Starting consumer %d success!\n",i);
    }

    for(i = 0; i < PRODUCER_NUM; ++i)
    {
        res = pthread_join(producer_thread[i],&thread_result);
        if(res != 0)
        {
            perror("thread producer failed");
            exit(EXIT_FAILURE);
        }
        printf("thread producer %d joined\n", i);
    }

    for(i = 0; i < PRODUCER_NUM; ++i)
    {
        res = pthread_join(consumer_thread[i],&thread_result);
        if(res != 0)
        {
            perror("thread consumer failed");
            exit(EXIT_FAILURE);
        }
        printf("thread consumer %d joined\n",i);
    }

    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&condCon);
    pthread_cond_destroy(&condPro);

    return 0;
}

void *producer_function(void *args)
{
    int index = *(int*)args;
    while(1){
        pthread_mutex_lock(&mutex);
        while(g_count >= MAXCAPACITY){
            printf("产品已满，生产者%d等待中\n", index);
            pthread_cond_wait(&condCon, &mutex);
        }
```

```
        ++g_count;
        printf("生产者%d生产了1个产品，现在共有%d个产品\n", index, g_count);
        pthread_cond_signal(&condPro);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}

void *consumer_function(void *args)
{
    int index = *(int*)args;
    while(1){
        pthread_mutex_lock(&mutex);
        while(g_count == 0){
            printf("缓冲区里没有产品了，消费者%d等待中\n", index);
            pthread_cond_wait(&condPro, &mutex);
        }
        --g_count;
        printf("消费者%d消费1个产品，缓冲区现有%d个产品\n", index, g_count);
        pthread_cond_signal(&condCon);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}
```

参考文献

[1] 李洁, 张瑜慧. 信号量在生产者-消费者及其变形问题中的应用 [J]. 福建电脑, 2012(02):175-177.

[2] 李志民, 赵一丁, 底恒. 操作系统进程同步的教学实践 [C]// 计算机研究新进展（2010）——河南省计算机学会 2010 年学术年会论文集. 2010. 步的教学实践 [C] 计算机研究新进展（2010）——河南省计算机学会 2010 年学术年会论文集. 2010.

[3] 陈涛, 任海兰. 基于 Linux 的多线程池并发 Web 服务器设计 [J]. 电子设计工程, 2015(11):175-177.

[4] 李盼盼, 赵浩. 基于信号量机制的生产者消费者问题的分析 [J]. 无线互联科技, 2013(11):101-102.

本科生课内实践成绩评定表

班级：软件 1702　　姓名：刘佳迎　　学号：0121710880223

序号	评分项目	满分	实得分
1	学习态度认真、遵守纪律	10	
2	设计分析合理性	10	
3	设计方案正确性、可行性、创造性	20	
4	设计结果正确性	40	
5	设计报告的规范性	10	
6	设计验收	10	
		总得分/等级	
评语：			

注：最终成绩以五级分制记。优（90-100 分）、良（80-89 分）、中（70-79 分）、及格（60-69 分）、60 分以下为不及格

指导教师签名：

年　　月　　日