《操作系统》课第八次实验报告

学院:	软件学院
姓名:	杨万里
学号:	2013774
邮箱:	2013774@mail.nankai.edu.cn
时间:	2022/11/5

0. 开篇感言

进程/线程的通信真有魅力啊！

把实际生活中的问题映射到程序当中真有意思。

在上一次利用生产者消费者思路实现目录拷贝的基础上再解决生产者消费者问题更轻松一些。

但是生产者消费者问题可以看作实际生活中的问题，需要考虑更多特殊的情况，并且处理方式各不相同（不能简单地遇到生产者、消费者、产品总数、缓冲区大小为 0 的情况就退出），在考虑特殊情况并解决的过程中，我进一步体会到生产者消费者问题的魅力。

1. 实验题目

写一个 C 程序利用多线程实现生产者消费者问题

2. 实验目标

（1） 编写一个 C/C++程序

（2） 利用多线程实现生产者消费者问题

3. 原理方法及源代码

（1） 根据输入的**命令行参数**分别得到生产者、消费者数量、产品总数和缓冲区大小；

（2） 编写**产品结构体** item_st（包含产品编号、生产者编号、生产时间、产品信息）和**缓冲区组成单元的结构体** buffer_st（包含缓冲区编号、是否为空和其中的产品）。

程序中缓冲区以数组的方式实现。

（3） **共享变量**除了缓冲区之外，还有：

```
int now_produce_id = 0;  //表示当前已生产的产品序号
int now_consume_id = 0;  //表示当前已消费的产品序号
int Index = 0;  //缓冲区的指针，指向缓冲区最新产品
```

因此需要三个**互斥锁** mutex：

```
pthread_mutex_t mutex_pro_id = PTHREAD_MUTEX_INITIALIZER;  //产品id的互斥锁
pthread_mutex_t mutex_con_id = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_buffer = PTHREAD_MUTEX_INITIALIZER;  //缓冲区的互斥锁
```

mutex_pro_id 对应 now_produce_id，mutex_con_id 对应 now_consume_id，mutex_buffer 对应缓冲区 buffers 和标记 index。

同时还需要两个**信号量**：

```
sem_t sem_empty;
sem_t sem_full;
```

（4） 编写生产产品的函数：

```
int produce_item(struct item_st* item){
    pthread_mutex_lock(&mutex_pro_id);
    if(now_produce_id >= K){  //产品达到上限，线程会结束
        pthread_mutex_unlock(&mutex_pro_id);
        return 0;  //表示结束信号
    }
    else{
        now_produce_id += 1;
        item->id = now_produce_id;  //产品id
        printf("生产者%d开始生产产品%d\n", item->produce_id, item->id);
        pthread_mutex_unlock(&mutex_pro_id);
    }
    //随机sleep，模拟生产用时
    sleep((unsigned)(rand()%10));
    //记录生产时间
    time_t now;
    time(&now);
    item->timestamp_p = now;
    printf("生产者%d生产产品%d结束时间%d\n", item->produce_id, item->id,
(unsigned)(item->timestamp_p));
    return 1;
}
```

该函数接收一个 item 进行生产，同时返回一个 int 值（0/1），0 代表结束信号，1 代表继续信号。

首先该函数会访问临界资源 now_produce_id，查看已经生产的产品数量，

如果达到产品总数，该函数提前终止，并且传回信号 0，告诉生产者函数需要终止；如果还需要生产，则将 now_produce_id 加 1，释放临界资源之后开始生产（sleep），生产结束时，在 item 中记录产品生产时间。

（5） 编写生产者线程函数：

```
void* producer(void* param){
    int id = (int)(param);  //生产者编号
    struct item_st item;  //产品壳子
    item.produce_id = id;
    while (1)
    {
        int signal = produce_item(&item);
        if(signal == 0){
            printf("生产过程结束，%d号生产者已经退出!\n", id);
            return;  //线程结束
        }
        else{
            printf("%d号生产者线程准备进入临界区\n", id);
            sem_wait(&sem_empty);
            pthread_mutex_lock(&mutex_buffer);
            printf("%d号生产者线程已经进入临界区\n", id);
            Index += 1;
            printf("%d号生产者在缓冲区的%d号产品项存入产品%d\n", id, Index,
item.id);
            buffers[Index].item = item;
            buffers[Index].isempty = 0;
            pthread_mutex_unlock(&mutex_buffer);
            sem_post(&sem_full);
            printf("%d号生产者线程已经离开临界区\n", id);
        }
    }
}
```

需要传入生产者的编号（id），准备一个待生产的 item；

循环体中，生产者会先调用生产产品函数 produce_item，如果得知产品数目达标，生产者线程直接结束；否则会得到一个生产好的产品，生产者等待进入临界区（缓冲区），进入临界区之后，将生产好的产品放入缓冲区，随后释放临界资源并离开。

（6） 编写消费者线程函数：

```
void* consumer(void* param){
    int id = (int)param;
    struct item_st item;
    while(1){
        pthread_mutex_lock(&mutex_con_id);
        if(now_consume_id >= K)  //产品达到上限，线程会结束
            pthread_mutex_unlock(&mutex_con_id);
            printf("消费过程已经结束，%d号消费者已经退出!\n", id);
            return;  //表示结束
        }
        else{
            now_consume_id += 1;
            pthread_mutex_unlock(&mutex_con_id);
            printf("%d号消费者线程准备进入临界区\n", id);
            sem_wait(&sem_full);
            pthread_mutex_lock(&mutex_buffer);
            printf("%d号消费者线程已经进入临界区\n", id);
            item = buffers[Index].item;
            buffers[Index].isempty = 1;
            printf("%d号消费者线程从缓冲区的%d号产品项取走了产品%d\n", id,
Index, item.id);
            Index -= 1;
            pthread_mutex_unlock(&mutex_buffer);
            sem_post(&sem_empty);
            printf("%d号消费者线程已经离开临界区\n", id);
            consume_item(item, id);
        }
    }
}
```

该函数同样需要传入消费者编号（id），以及一个接收产品的 item。在循环体中，首先访问临界资源 now_consume_id，查看已经消费的产品是否到达上限，如果到达，消费者线程结束；否则将已经消费的产品数目加1，并且释放该临界资源。随后等待进入临界区（缓冲区），进入缓冲区之后，取下产品，释放临界资源。最后调用消费产品的函数 consum_item。

（7） 编写消费产品的函数：

```
void consume_item(struct item_st item, int id){
    printf("%d号消费者开始消费%d号生产者生产的%d号产品\n", id, item.produce_id,
item.id);
    sleep((unsigned)(rand()%10));
    //记录消费时间
    time_t now;
    time(&now);
    printf("%d号消费者消费产品%d结束时间%d\n", id, item.id, (unsigned)now);
}
```

该函数接收待消费的产品和消费者编号，消费（sleep）一段时间，打印消费结束时间即可。

（8） 编写主函数：

定义随机数种子

```
srand((unsigned)time(NULL));
```

接收命令行参数

```
I = atoi(argv[1]);
J = atoi(argv[2]);
K = atoi(argv[3]);
N = atoi(argv[4]);
```

初始化缓冲区

```
   buffers = malloc(sizeof(struct buffer_st) * (N + 1));  //创建大小为N的缓冲区
(0不要)
   //初始化
   for (int i = 1; i <= N; i++)
   {
       buffers[i].id = i;
       buffers[i].isempty = 1;
   }
```

初始化信号量、互斥锁

```
sem_init(&sem_empty, 0, N);
sem_init(&sem_full, 0, 0);

pthread_mutex_init(&mutex_buffer, 0);
pthread_mutex_init(&mutex_pro_id, 0);
pthread_mutex_init(&mutex_con_id, 0);
```

创建生产者和消费者线程

```
pthread_t produce_threads[I];
pthread_t consume_threads[J];

for(int i=0;i<I;i++){
  pthread_create(&produce_threads[i], NULL, producer, (void*)(i+1));
}
for(int i=0;i<J;i++){
  pthread_create(&consume_threads[i], NULL, consumer, (void*)(i+1));
}
```

设置屏障，统一多线程步伐

```
for(int i=0;i<I;i++){
  pthread_join(produce_threads[i], 0);
}
for(int i=0;i<J;i++){
  pthread_join(consume_threads[i], 0);
}
```

销毁信号量和互斥锁

```
pthread_mutex_destroy(&mutex_buffer);
pthread_mutex_destroy(&mutex_con_id);
pthread_mutex_destroy(&mutex_pro_id);

sem_destroy(&sem_full);
sem_destroy(&sem_empty);
```

（9） 考虑特殊情况：

1）需要生产的产品总数 K=0：

这种情况生产者和消费者线程都能在开始就正常退出，无需额外的修改。

2）缓冲区大小 N=0：

缓冲区大小为 0，则代表无法放入任何产品，也无法消费任何产品，因此程序需要直接退出。

```
if(N == 0){
  printf("缓冲区大小为0，程序无法正常运行，故退出！\n");
}
```

3）生产者数目 I=0，消费者数目 J!=0：

这种情况不启动生产者线程，无法生产产品，消费者线程会一直阻塞。因此在**消费者线程函数的开始加上判断**，如果 I==0，则直接结束线程。

```
void* consumer(void* param){
    int id = (int)param;
    if(I == 0){
      printf("检测到当前程序没有生产者，%d号消费者线程退出!\n", id);
      return;
    }
```

4）生产者数目 I!=0，消费者数目 J=0：

如果生产者数目不为 0，消费者数目为 0，理想的情况不是程序直接退出，而是**生产完所有产品或者用产品填满缓冲区之后**，消费者线程再退出。

因此，如果需要生产的**产品总数 K<=缓冲区大小 N** 时，生产者线程能正常运行；如果**产品总数 K>缓冲区大小 N** 时，按照当前程序，生产者线程会一直阻塞，因此需要**把产品总数 K 修改为 N 的值**，生产者生产的产品填满缓冲区之后，由于**不会有消费者消费产品**，生产者线程退出。

在主函数开始进行修改

```
    if(I != 0 && J == 0){
        if(K > N){
            K = N;
        }
    }
```

5）生产者数目 I=0，消费者数目 J=0：

既不启动生产者线程，又不启动消费者线程，程序正常结束，无需额外修改。

以上则完成了整个程序的编写，**完整的源代码附在实验报告结尾处**。

4. 具体步骤

（1） 按照上一小节的原理方法编写程序

（2） 编译程序

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab8_code$ gcc pc.c -o pc -lpthread
```

（3） 利用测试样例验证程序的正确性：

a）2 个生产者；3 个消费者；15 个产品；缓冲区大小为 4

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab8_code$ ./pc 2 3 15 4
生产者消费者程序开始!
生产者1开始生产产品1
1号消费者线程准备进入临界区
生产者2开始生产产品2
2号消费者线程准备进入临界区
3号消费者线程准备进入临界区
生产者2生产产品2结束时间1667611516
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品2
2号生产者线程已经离开临界区
生产者2开始生产产品3
1号消费者线程已经进入临界区
1号消费者线程从缓冲区的1号产品项取走了产品2
1号消费者线程已经离开临界区
1号消费者开始消费2号生产者生产的2号产品
生产者1生产产品1结束时间1667611517
1号生产者线程准备进入临界区
1号生产者线程已经进入临界区
1号生产者在缓冲区的1号产品项存入产品1
1号生产者线程已经离开临界区
生产者1开始生产产品4
2号消费者线程已经进入临界区
2号消费者线程从缓冲区的1号产品项取走了产品1
2号消费者线程已经离开临界区
2号消费者开始消费1号生产者生产的1号产品
2号消费者消费产品1结束时间1667611518
2号消费者线程准备进入临界区
生产者2生产产品3结束时间1667611522
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品3
2号生产者线程已经离开临界区
生产者2开始生产产品5
3号消费者线程已经进入临界区
3号消费者线程从缓冲区的1号产品项取走了产品3
3号消费者线程已经离开临界区
3号消费者开始消费2号生产者生产的3号产品
生产者2生产产品5结束时间1667611522
```

```
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品5
2号生产者线程已经离开临界区
生产者2开始生产产品6
2号消费者线程已经进入临界区
2号消费者线程从缓冲区的1号产品项取走了产品5
2号消费者线程已经离开临界区
2号消费者开始消费2号生产者生产的5号产品
1号消费者消费产品2结束时间1667611523
1号消费者线程准备进入临界区
生产者2生产产品6结束时间1667611523
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品6
2号生产者线程已经离开临界区
生产者2开始生产产品7
1号消费者线程已经进入临界区
1号消费者线程从缓冲区的1号产品项取走了产品6
1号消费者线程已经离开临界区
1号消费者开始消费2号生产者生产的6号产品
生产者1生产产品4结束时间1667611525
1号生产者线程准备进入临界区
1号生产者线程已经进入临界区
1号生产者在缓冲区的1号产品项存入产品4
1号生产者线程已经离开临界区
生产者1开始生产产品8
3号消费者消费产品3结束时间1667611526
3号消费者线程准备进入临界区
3号消费者线程已经进入临界区
3号消费者线程从缓冲区的1号产品项取走了产品4
3号消费者线程已经离开临界区
3号消费者开始消费1号生产者生产的4号产品
生产者2生产产品7结束时间1667611527
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品7
2号生产者线程已经离开临界区
生产者2开始生产产品9
```

```
2号消费者消费产品5结束时间1667611527
2号消费者线程准备进入临界区
2号消费者线程已经进入临界区
2号消费者线程从缓冲区的1号产品项取走了产品7
2号消费者线程已经离开临界区
2号消费者开始消费2号生产者生产的7号产品
1号消费者消费产品6结束时间1667611529
1号消费者线程准备进入临界区
生产者1生产产品8结束时间1667611531
1号生产者线程准备进入临界区
1号生产者线程已经进入临界区
1号生产者在缓冲区的1号产品项存入产品8
1号生产者线程已经离开临界区
生产者1开始生产产品10
1号消费者线程已经进入临界区
1号消费者线程从缓冲区的1号产品项取走了产品8
1号消费者线程已经离开临界区
1号消费者开始消费1号生产者生产的8号产品
生产者1生产产品10结束时间1667611534
1号生产者线程准备进入临界区
1号生产者线程已经进入临界区
1号生产者在缓冲区的1号产品项存入产品10
1号生产者线程已经离开临界区
生产者1开始生产产品11
3号消费者消费产品4结束时间1667611534
3号消费者线程准备进入临界区
3号消费者线程已经进入临界区
3号消费者线程从缓冲区的1号产品项取走了产品10
3号消费者线程已经离开临界区
3号消费者开始消费1号生产者生产的10号产品
生产者2生产产品9结束时间1667611536
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品9
2号生产者线程已经离开临界区
生产者2开始生产产品12
2号消费者消费产品7结束时间1667611536
2号消费者线程准备进入临界区
2号消费者线程已经进入临界区
2号消费者线程从缓冲区的1号产品项取走了产品9
```

```
2号消费者线程已经离开临界区
2号消费者开始消费2号生产者生产的9号产品
生产者2生产产品12结束时间1667611536
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品12
2号生产者线程已经离开临界区
生产者2开始生产产品13
1号消费者消费产品8结束时间1667611537
1号消费者线程准备进入临界区
1号消费者线程已经进入临界区
1号消费者线程从缓冲区的1号产品项取走了产品12
1号消费者线程已经离开临界区
1号消费者开始消费2号生产者生产的12号产品
3号消费者消费产品10结束时间1667611538
3号消费者线程准备进入临界区
生产者2生产产品13结束时间1667611539
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品13
2号生产者线程已经离开临界区
生产者2开始生产产品14
3号消费者线程已经进入临界区
3号消费者线程从缓冲区的1号产品项取走了产品13
3号消费者线程已经离开临界区
3号消费者开始消费2号生产者生产的13号产品
生产者2生产产品14结束时间1667611540
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品14
2号生产者线程已经离开临界区
生产者2开始生产产品15
1号消费者消费产品12结束时间1667611541
1号消费者线程准备进入临界区
1号消费者线程已经进入临界区
1号消费者线程从缓冲区的1号产品项取走了产品14
1号消费者线程已经离开临界区
1号消费者开始消费2号生产者生产的14号产品
生产者1生产产品11结束时间1667611543
```

```
1号生产者线程准备进入临界区
1号生产者线程已经进入临界区
1号生产者在缓冲区的1号产品项存入产品11
1号生产者线程已经离开临界区
生产过程结束，1号生产者已经退出!
2号消费者消费产品9结束时间1667611544
2号消费者线程准备进入临界区
2号消费者线程已经进入临界区
2号消费者线程从缓冲区的1号产品项取走了产品11
2号消费者线程已经离开临界区
2号消费者开始消费1号生产者生产的11号产品
2号消费者消费产品11结束时间1667611544
2号消费者线程准备进入临界区
3号消费者消费产品13结束时间1667611544
消费过程已经结束，3号消费者已经退出!
生产者2生产产品15结束时间1667611544
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品15
2号生产者线程已经离开临界区
生产过程结束，2号生产者已经退出!
2号消费者线程已经进入临界区
2号消费者线程从缓冲区的1号产品项取走了产品15
2号消费者线程已经离开临界区
2号消费者开始消费2号生产者生产的15号产品
1号消费者消费产品14结束时间1667611547
消费过程已经结束，1号消费者已经退出!
2号消费者消费产品15结束时间1667611550
消费过程已经结束，2号消费者已经退出!
生产者消费者程序结束!
```

b）0 个生产者；3 个消费者；15 个产品；缓冲区大小为 4

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab8_code$ ./pc 0 3 15 4
生产者消费者程序开始!
检测到当前程序没有生产者，1号消费者线程退出!
检测到当前程序没有生产者，2号消费者线程退出!
检测到当前程序没有生产者，3号消费者线程退出!
生产者消费者程序结束!
```

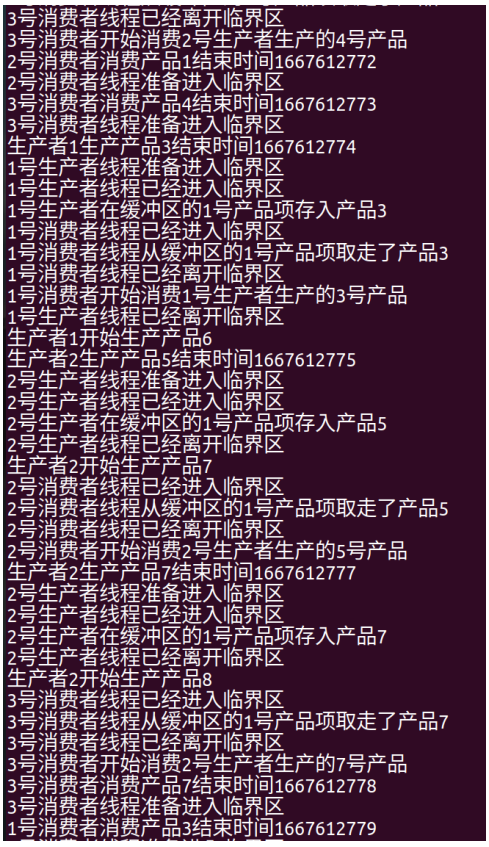
c）2 个生产者；0 个消费者；15 个产品；缓冲区大小为 4

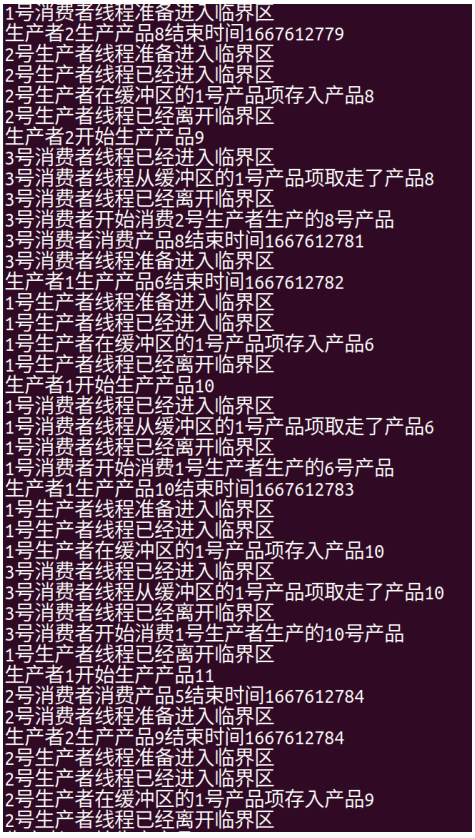
```
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab8_code$ ./pc 2 0 15 4
生产者消费者程序开始!
生产者1开始生产产品1
生产者2开始生产产品2
生产者2生产产品2结束时间1667612552
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品2
2号生产者线程已经离开临界区
生产者2开始生产产品3
生产者1生产产品1结束时间1667612559
1号生产者线程准备进入临界区
1号生产者线程已经进入临界区
1号生产者在缓冲区的2号产品项存入产品1
1号生产者线程已经离开临界区
生产者1开始生产产品4
生产者1生产产品4结束时间1667612560
1号生产者线程准备进入临界区
1号生产者线程已经进入临界区
1号生产者在缓冲区的3号产品项存入产品4
1号生产者线程已经离开临界区
生产过程结束，1号生产者已经退出!
生产者2生产产品3结束时间1667612560
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的4号产品项存入产品3
2号生产者线程已经离开临界区
生产过程结束，2号生产者已经退出!
生产者消费者程序结束!
```

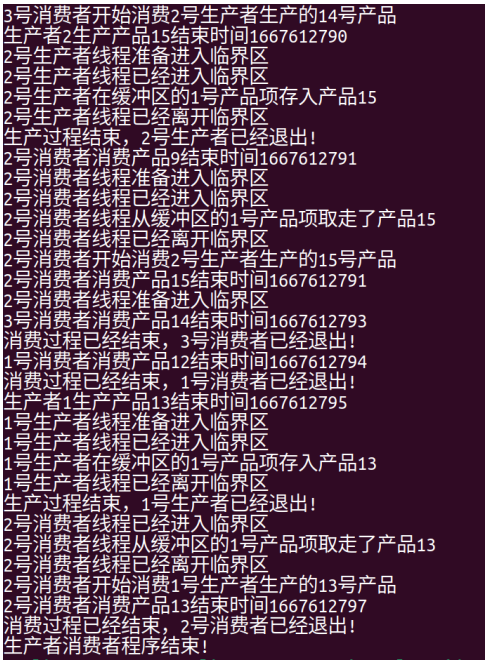
由于没有消费者，产品总数为 15，大于缓冲区大小 4，因此 2 个生产者只需要在产品填满缓冲区（4）之后退出即可。

d）2 个生产者；3 个消费者；15 个产品；缓冲区大小为 1

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab8_code$ ./pc 2 3 15 1
生产者消费者程序开始!
生产者2开始生产产品1
生产者1开始生产产品2
1号消费者线程准备进入临界区
2号消费者线程准备进入临界区
3号消费者线程准备进入临界区
生产者1生产产品2结束时间1667612768
1号生产者线程准备进入临界区
1号生产者线程已经进入临界区
1号生产者在缓冲区的1号产品项存入产品2
1号消费者线程已经进入临界区
1号消费者线程从缓冲区的1号产品项取走了产品2
1号消费者线程已经离开临界区
1号消费者开始消费1号生产者生产的2号产品
1号消费者消费产品2结束时间1667612768
1号消费者线程准备进入临界区
1号生产者线程已经离开临界区
生产者1开始生产产品3
生产者2生产产品1结束时间1667612770
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品1
2号生产者线程已经离开临界区
生产者2开始生产产品4
2号消费者线程已经进入临界区
2号消费者线程从缓冲区的1号产品项取走了产品1
2号消费者线程已经离开临界区
2号消费者开始消费2号生产者生产的1号产品
生产者2生产产品4结束时间1667612772
2号生产者线程准备进入临界区
2号生产者线程已经进入临界区
2号生产者在缓冲区的1号产品项存入产品4
2号生产者线程已经离开临界区
生产者2开始生产产品5
3号消费者线程已经进入临界区
3号消费者线程从缓冲区的1号产品项取走了产品4
```







缓冲区大小为 1，生产者需要等消费者将缓冲区中已有的 1 个产品取走之后才可以继续放入产品，因此程序整体会比缓冲区大小为 4 的程序更慢。

e）0 个生产者；0 个消费者；15 个产品；缓冲区大小为 4

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab8_code$ ./pc 0 0 15 4
生产者消费者程序开始!
生产者消费者程序结束!
```

由于没有生产者和消费者，程序开始之后就退出。

f）2 个生产者；3 个消费者；0 个产品；缓冲区大小为 4

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab8_code$ ./pc 2 3 0 4
生产者消费者程序开始!
生产过程结束，2号生产者已经退出!
生产过程结束，1号生产者已经退出!
消费过程已经结束，3号消费者已经退出!
消费过程已经结束，2号消费者已经退出!
消费过程已经结束，1号消费者已经退出!
生产者消费者程序结束!
```

产品总数 K=0，没有生产和消费任务，虽然创建了生产者和消费者线程，但是二者都很快就结束了。

至此，**所有测试样例均通过**，根据输出结果可以判断程序**没有产生竞态和死锁**，并且针对特殊情况也能正常运行，实验成功！

5. 总结心得

（1） 通过本次实验，我进一步加深了对生产者消费者问题的理解，对于线程间的通信与协作有了更深刻的认知。

（2） 经过上次实验和本次实验的实践，我更理解了信号量与互斥锁避免竞态的原理，也更理解了李老师课上说的“其实这几种方法本质是一样的，只是应用的场景不同”。

（3） 上一次实验只是利用生产者消费者思路解决了拷贝目录的问题，这一次实际解决生产者消费者问题，需要考虑更多实际生活中可能产生的问题，比如有消费者没有生产者、缓冲区大小为 0 等等情况，在发现问题解决问题的过程中，完善了程序逻辑。

6. 参考资料

实验指导书《lab_IPC_PC.pdf》

7. 实验源代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

```
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>
#include <semaphore.h>
#include <time.h>

int I;  //生产者数量
int J;  //消费者数量
int K;  //产品总数
int N;  //缓冲区大小

int now_produce_id = 0;  //表示当前已生产的产品序号
int now_consume_id = 0;  //表示当前已消费的产品序号
int Index = 0;  //缓冲区的指针，指向缓冲区最新产品

pthread_mutex_t mutex_pro_id = PTHREAD_MUTEX_INITIALIZER;  //产品 id 的互斥锁
pthread_mutex_t mutex_con_id = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex_buffer = PTHREAD_MUTEX_INITIALIZER;  //缓冲区的互斥锁

sem_t sem_empty;
sem_t sem_full;

//产品结构体
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
};

//缓冲区结构体
struct buffer_st
{
    int id;  //缓冲区编号
    int isempty;  //是否为空,1 空 0 非空
    struct item_st item;  //具体产品
};

struct buffer_st* buffers;

```

```
48. //生产产品函数
49. int produce_item(struct item_st* item){
50.     pthread_mutex_lock(&mutex_pro_id);
51.     if(now_produce_id >= K){  //产品达到上限，线程会结束
52.         pthread_mutex_unlock(&mutex_pro_id);
53.         return 0;  //表示结束信号
54.     }
55.     else{
56.         now_produce_id += 1;
57.         item->id = now_produce_id;  //产品 id
58.         printf("生产者%d 开始生产产
    品%d\n", item->produce_id, item->id);
59.         pthread_mutex_unlock(&mutex_pro_id);
60.     }
61.     //随机 sleep，模拟生产用时
62.     sleep((unsigned)(rand()%10));
63.     //记录生产时间
64.     time_t now;
65.     time(&now);
66.     item->timestamp_p = now;
67.     printf("生产者%d 生产产品%d 结束时
    间%d\n", item->produce_id, item->id, (unsigned)(item->timestamp_p));
68.     return 1;
69. }
70.
71. //生产者线程函数
72. void* producer(void* param){
73.     int id = (int)(param);  //生产者编号
74.     struct item_st item;  //产品壳子
75.     item.produce_id = id;
76.     while (1)
77.     {
78.         int signal = produce_item(&item);
79.         if(signal == 0){
80.             printf("生产过程结束，%d 号生产者已经退出!\n", id);
81.             return;  //线程结束
82.         }
83.         else{
84.             printf("%d 号生产者线程准备进入临界区\n", id);
85.             sem_wait(&sem_empty);
86.             pthread_mutex_lock(&mutex_buffer);
87.             printf("%d 号生产者线程已经进入临界区\n", id);
88.             Index += 1;
```

```
89.             printf("%d 号生产者在缓冲区的%d 号产品项存入产
    品%d\n", id, Index, item.id);
90.             buffers[Index].item = item;
91.             buffers[Index].isempty = 0;
92.             pthread_mutex_unlock(&mutex_buffer);
93.             sem_post(&sem_full);
94.             printf("%d 号生产者线程已经离开临界区\n", id);
95.         }
96.     }
97. }
98.
99. //消费者线程函数
100.   void* consumer(void* param){
101.       int id = (int)param;
102.       if(I == 0){
103.         printf("检测到当前程序没有生产者，%d 号消费者线程退出!\n", id);
104.       }
105.       struct item_st item;
106.       while(1){
107.           pthread_mutex_lock(&mutex_con_id);
108.           if(now_consume_id >= K){  //产品达到上限，线程会结束
109.               pthread_mutex_unlock(&mutex_con_id);
110.               printf("消费过程已经结束，%d 号消费者已经退出!\n", id);
111.               return;  //表示结束
112.           }
113.           else{
114.               now_consume_id += 1;
115.               pthread_mutex_unlock(&mutex_con_id);
116.               printf("%d 号消费者线程准备进入临界区\n", id);
117.               sem_wait(&sem_full);
118.               pthread_mutex_lock(&mutex_buffer);
119.               printf("%d 号消费者线程已经进入临界区\n", id);
120.               item = buffers[Index].item;
121.               buffers[Index].isempty = 1;
122.               printf("%d 号消费者线程从缓冲区的%d 号产品项取走了产
    品%d\n", id, Index, item.id);
123.               Index -= 1;
124.               pthread_mutex_unlock(&mutex_buffer);
125.               sem_post(&sem_empty);
126.               printf("%d 号消费者线程已经离开临界区\n", id);
127.               consume_item(item, id);
128.           }
129.       }
130.   }
```

```
131.
132.   //消费函数
133.   void consume_item(struct item_st item, int id){
134.       printf("%d 号消费者开始消费%d 号生产者生产的%d 号产品
    \n", id, item.produce_id, item.id);
135.       sleep((unsigned)(rand()%10));
136.       //记录消费时间
137.       time_t now;
138.       time(&now);
139.       printf("%d 号消费者消费产品%d 结束时
    间%d\n", id, item.id, (unsigned)now);
140.   }
141.
142.   int main(int argc, char *argv[]){
143.       printf("生产者消费者程序开始!\n");
144.
145.       srand((unsigned)time(NULL));
146.
147.       //输入基本参数
148.       I = atoi(argv[1]);
149.       J = atoi(argv[2]);
150.       K = atoi(argv[3]);
151.       N = atoi(argv[4]);
152.
153.       if(N == 0){
154.         printf("缓冲区大小为 0，程序无法正常运行，故退出！ \n");
155.       }
156.       if(I != 0 && J == 0){
157.           if(K > N){
158.              K = N;
159.           }
160.       }
161.
162.       buffers = malloc(sizeof(struct buffer_st) * (N + 1));  //创建大
    小为 N 的缓冲区（0 不要）
163.       //初始化
164.       for (int i = 1; i <= N; i++)
165.       {
166.           buffers[i].id = i;
167.           buffers[i].isempty = 1;
168.       }
169.
170.       sem_init(&sem_empty, 0, N);
171.       sem_init(&sem_full, 0, 0);
```

```
172.
173.        pthread_mutex_init(&mutex_buffer, 0);
174.        pthread_mutex_init(&mutex_pro_id, 0);
175.        pthread_mutex_init(&mutex_con_id, 0);
176.
177.        pthread_t produce_threads[I];
178.        pthread_t consume_threads[J];
179.
180.        for(int i=0;i<I;i++){
181.          pthread_create(&produce_threads[i], NULL, producer, (void*)(
    i+1));
182.        }
183.        for(int i=0;i<J;i++){
184.          pthread_create(&consume_threads[i], NULL, consumer, (void*)(
    i+1));
185.        }
186.
187.        for(int i=0;i<I;i++){
188.          pthread_join(produce_threads[i], 0);
189.        }
190.        for(int i=0;i<J;i++){
191.          pthread_join(consume_threads[i], 0);
192.        }
193.
194.        pthread_mutex_destroy(&mutex_buffer);
195.        pthread_mutex_destroy(&mutex_con_id);
196.        pthread_mutex_destroy(&mutex_pro_id);
197.
198.        sem_destroy(&sem_full);
199.        sem_destroy(&sem_empty);
200.
201.        printf("生产者消费者程序结束!\n");
202.        return 0;
203.    }
```