实验一　P、V 原语的模拟实现　学时:2

（一）实验类型：综合型

（二）实验类别：专业实验

（三）实验要求：必修

（四）实验目的

1.掌握临界区的概念及临界区的设计原则；

2.掌握信号量的概念、wait\signal 操作的含义以及应用 wait\signal 操作实现进程的同步与互斥；

3.分析进程争用资源的现象，学习解决进程互斥的方法。

（五）实验内容

分析进程的同步与互斥现象，编程实现经典的进程同步问题——生产者消费者问题的模拟

生产者--消费者问题表述：

有一环形缓冲池，包含 n 个缓冲区（0~n-1）。

有两类进程：一组生产者进程和一组消费者进程，生产者进程向空的缓冲区中放产品，消费者进程从满的缓冲区中取走产品。

所有进程必须对缓冲区进行互斥的访问。

生产者不能向满缓冲区写数据，消费者不能从空缓冲区取数据，即生产者与消费者必须同步。

计算机系统中对资源的分配与释放过程：计算机系统中的每个进程都可以消费或生产某类资源。当系统中某一进程使用某一资源时，可以看作是消耗，且该进程称为消费者。而当某个进程释放资源时，则它就相当一个生产者。

定义生产者消费者问题中的各数据结构，并初始化。

信号量，初值。

编写 wait signal 操作。

编写生产者与消费者程序，利用信号量及其 PV 操作，实现生产者与消费者之间的同步与互斥。

模拟显示生产者与消费者同步与互斥的效果。

（六）实验方法、步骤及结果测试

2. 利用记录型信号量实现生产者—消费者问题

(1) 实现 1 个生产者—1 个消费者模型。条件限制：若容器已满，生产者不能生产，待消费者消费；若产品为空，则消费者不能消费，需等待生产者生产；生产者循环生产，消费者循环消费。

参考代码如下：

```
//pthread_sem_app1.c
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>
#define NUM 5
int queue[NUM];                          //全局数组实现环形队列
int in=0;
int out=0;
sem_t blank_number, product_number;      //空格子信号量，产品信号量
sem_t mutex;                             //互斥锁
void *producer(void *arg)
{
    int j=1;
    while (j<=6) {                       //生产者循环生产
        sem_wait(&blank_number);         //生产者将空格子数减 1，为 0 则阻塞等待
        sem_wait(&mutex);
        queue[in] = rand() % 1000 + 1;   //生产一个产品
        printf("----Produce---%d\n", queue[in]);
        in= (in + 1) % NUM;              //借助下标实现环形
```

```
        sem_post(&mutex);
        sem_post(&product_number);                    //将产品数加 1
        sleep(rand() % 2);
        j++;
    }
    pthread_exit(NULL);
}
void *consumer(void *arg)
{
    int j=1;
    while (j<=6) {                                    //消费者循环消费
        sem_wait(&product_number);                    //消费者将产品数减 1，为 0 则阻塞等待
        sem_wait(&mutex);
        printf("-Consume---%d t%lu\n", queue[out], pthread_self());
        queue[out] = 0;                               //消费一个产品
        out = (out + 1) % NUM;
        sem_post(&mutex);
        sem_post(&blank_number);                      //消费掉以后，将空格子数加 1
        sleep(rand() % 1);
        j++;
    }
    pthread_exit(NULL);
}
int main(int argc, char *argv[])
{
    pthread_t pid, cid1,cid2;
    int ret;
    sem_init(&blank_number, 0, NUM);                  //初始化空格子信号量为 5
    sem_init(&product_number, 0, 0);                  //初始化产品数信号量为 0
    sem_init(&mutex,0,1);                             //互斥锁初始值为 1
    pthread_create(&pid, NULL, producer, NULL);
    pthread_create(&cid1, NULL, consumer, NULL);
    pthread_join(pid, NULL);
    pthread_join(cid1, NULL);
    sem_destroy(&blank_number);
    sem_destroy(&product_number);
    sem_destroy(&mutex);
    return 0;
}
```

测试结果如下：

```
[root@localhost chaper2]# gcc pthread_sem_app1.c -lpthread
[root@localhost chaper2]# ./a.out
----Produce---384
----Produce---778
-Consume---384   140070862141184
-Consume---778   140070862141184
----Produce---387
-Consume---387   140070862141184
----Produce---422
----Produce---28
----Produce---60
-Consume---422   140070862141184
-Consume---28    140070862141184
-Consume---60    140070862141184
[root@localhost chaper2]#
```

(2) NUM 个生产者—NUM 个消费者模型：每个生产者生产 1 个产品，每个消费者消费 1 个产品；若容器已满，生产者不能生产，需等待消费者消费；若产品为空，则消费者不能消费，需等待生产者生产。

参考代码如下：

```
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>
#define NUM 10                          // NUM 为 10，即 10 个生产者和 10 个消费者共享循环队列
int queue[NUM];                         //全局数组实现环形队列
int in=0;
int out=0;
sem_t blank_number, product_number;     //空格子信号量，产品信号量
sem_t mutex;                            //互斥锁

void *producer(void *arg)
{
    sem_wait(&blank_number);            //生产者将空格子数减 1，为 0 则阻塞等待
    sem_wait(&mutex);
    queue[in] = rand() % 1000 + 1;      //生产一个产品
    printf("----Produce---%d\n", queue[in]);
    in = (in + 1)% NUM;                 //借助下标实现环形
```

```
    sem_post(&mutex);
    sem_post(&product_number);                    //将产品数加 1
    sleep(rand() % 1);
    pthread_exit(NULL);
}
void *consumer(void *arg)
{
    sem_wait(&product_number);                    //消费者将产品数减 1，为 0 则阻塞等待
    sem_wait(&mutex);
    printf("-Consume---%d t%lu\n", queue[out], pthread_self());
    queue[out] = 0;                               //消费一个产品
    out = (out + 1) % NUM;
    sem_post(&mutex);
    sem_post(&blank_number);                      //消费掉以后，将空格子数加 1
    sleep(rand() % 1);
    pthread_exit(NULL);
}
int main(int argc, char *argv[])
{
    int i,j;                                      //创建生产者和消费者数量
    pthread_t pid[NUM], cid[NUM];
    int ret;
    sem_init(&blank_number, 0, NUM);              //初始化空格子信号量为 NUM
    sem_init(&product_number, 0, 0);              //初始化产品数信号量为 0
    sem_init(&mutex, 0, 1);                       //互斥锁初始值为 1
    for(i=0;i<NUM;i++)                            //创建 NUM 个生产者
    {
        pthread_create(&pid[i], NULL, producer, NULL);
    }
    for(i=0;i<NUM;i++)                            //创建 NUM 个消费者
    {
        pthread_create(&cid[i], NULL, consumer, NULL);
    }
    for(i=0; i<NUM; i++)                          //回收 NUM 个生产者
    {
        pthread_join(pid[i], NULL);
    }
    for(i=0; i<NUM; i++)                          //回收 NUM 个生产者
    {
```

```
        pthread_join(cid[i], NULL);
    }
    sem_destroy(&blank_number);
    sem_destroy(&product_number);
    sem_destroy(&mutex);
    return 0;
}
```

测试结果如下：

```
[root@localhost chaper2]# ./a.out
----Produce---384
----Produce---778
----Produce---794
----Produce---387
----Produce---650
----Produce---363
----Produce---691
----Produce---764
-Consume---384   139841141847808
----Produce---427
-Consume---778   139841131357952
-Consume---794   139841120868096
-Consume---387   139841110378240
-Consume---650   139841099888384
-Consume---363   139841089398528
-Consume---691   139841078908672
-Consume---764   139841068418816
-Consume---427   139841057928960
----Produce---124
-Consume---124   139841047439104
[root@localhost chaper2]#
```

【思考练习】

1. 有一个计算进程和一个打印进程，它们共享一个单缓冲区，计算进程
一个整型结果并将该结果放入单缓冲区，打印进程则负责从单缓冲区中取出
行打印。试用信号量来实现它们的同步关系。

2. 有三个进程 PA、PB 和 PC 协作解决文件打印问题。PA 将文件记录从
的缓冲区 1，每执行一次读一个记录；PB 将缓冲区 1 的内容复制到缓冲区 2
次复制一个记录；PC 将缓冲区 2 的内容打印出来，每执行一次打印一个记
大小与记录大小一样。试用信号量来保证文件的正确打印。

（七）实验总结