

515030910605

李洋

2017年11月20日

# Project4

## Producer-Consumer Problem

### 1. Introduction

In this experiment, we will design a programming solution to the bounded-buffer problem using the producer and consumer processes .

These are the things I have done in the code file:

- a) Define producer and consumer function to add data or consume data of buffer.
- b) Define and use semaphore to keep process in sync.
- c) Create thread to execute the producer and consumer function.

I will introduce my method in the next part.

### 2. Steps

#### 2.1. Workflow

- a) Accept the inputs and check them.
- b) Add semaphores to keep threads in sync.
- c) Create threads to execute the producers and consumers threads.
- d) Display the result.

## 2.2. Accept and check the inputs

We accept 3 parameters, sleep time, the number of producers and the number of consumers. We first check whether the number of parameters is right. Then we assign the parameters to 3 variables. In particular, we can't create producers or consumers infinitely. So we have a bound — `THREAD_NUM`. If the number of producers or the number of consumers more than it, we will return error information.

```
/* check the number of parameters*/
if(argc != 4){
    printf("Wrong number of parameters\n");
    return 0;
}

int sleeptime = atoi(argv[1]);
int pNum = atoi(argv[2]);
int cNum = atoi(argv[3]);

/* The number of producers or consumers no more than THREAD_NUM*/
if (pNum > THREAD_NUM || cNum > THREAD_NUM){
    printf("The number of producers or consumers no more than %d\n", THRE
    return 0;
}
```

## 2.3. Add semaphores to keep threads in sync

The key point of keep threads in sync is semaphores. We create three semaphores to solve the Producer-Consumer Problem.

We create 3 semaphores: mutex, full and empty. We initialize mutex with 1, full with 0 and empty with `BUFFER_SIZE`.

Then we write producer and consumer functions. In producer function, we wait(empty) to wait for a empty buffer and wait(mutex) to

wait for a free control of buffer. Then producer execute insert\_item function to add data to buffer. In the end, we release the semaphores.

As for consumer function, we wait(full) to wait for a full buffer and wait(mutex) to wait for a free control of buffer. Then consumer execute remove\_item function to get data to buffer and release buffer. In the end, we release the semaphores.

```
/* define the producer and consumer*/
void *producer(void *param){
    buffer_item ran;

    while(1){
        /* sleep for a random period of time */
        sleep(rand()%6);
        /* generate a random number */
        ran = rand()%30;

        sem_wait(&empty);
        sem_wait(&mutex);

        /* add nextpt to buffer*/
        printf("producer produced %d\n", ran);
        if( insert_item(ran))
            printf("report error condition");

        sem_post(&mutex);
        sem_post(&full);
    }
}

void *consumer(void *param){
    buffer_item ran;

    while(1){
        /* sleep for a random period of time */
        sleep(rand()%6);

        sem_wait(&full);
        sem_wait(&mutex);

        int re = remove_item(&ran);

        sem_post(&mutex);
        sem_post(&empty);

        if(re)
            printf("report error condition");
        else
            printf("consumer consumed %d\n", ran);
    }
}
```

## 2.4. Create threads to execute the producers and consumers

This operation are executed in main() function.

We create two arrays to store producer threads and consumer threads. And we also initialize the semaphores.

Then we create threads to execute the function just like what we do in the last project.

```
/* Store the threads*/
pthread_t producers_thread[THREAD_NUM];
pthread_t consumers_thread[THREAD_NUM];

/* initialize semaphore*/
sem_init(&mutex, 0, 1);
sem_init(&empty, 0, BUFFER_SIZE);
sem_init(&full, 0, 0);
srand(time(NULL));

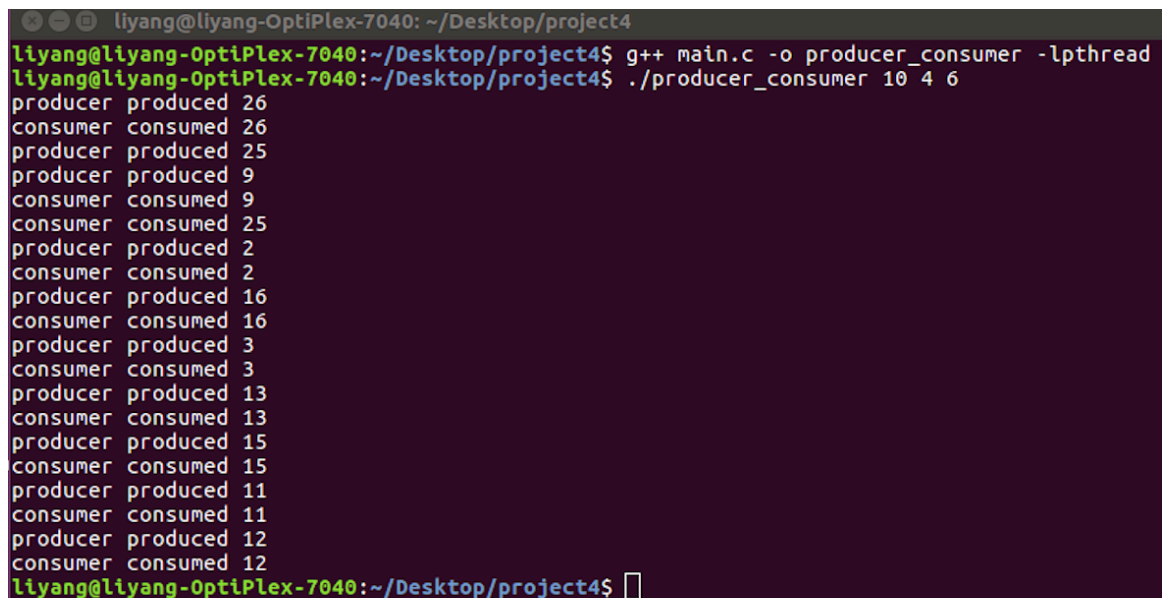
/* create threads */
for (int i = 0; i < pNum; ++i)
{
    if(pthread_create(&producers_thread[i], NULL, producer, NULL))
        printf("Producer Thread Creating Error!\n");
}

for (int i = 0; i < cNum; ++i)
{
    if(pthread_create(&consumers_thread[i], NULL, consumer, NULL))
        printf("consumer Thread Creating Error!\n");
}
```

### 3. Test result show

We use `./producer_consumer [sleeptime] [number of producers]`  
`[number of consumers]` “ to show the solution of producer-consumer problem.

We can see that producers and consumers add and use data from buffer alternatively in sleep time before `main()` terminate.



```
liyang@liyang-OptiPlex-7040: ~/Desktop/project4
liyang@liyang-OptiPlex-7040:~/Desktop/project4$ g++ main.c -o producer_consumer -lpthread
liyang@liyang-OptiPlex-7040:~/Desktop/project4$ ./producer_consumer 10 4 6
producer produced 26
consumer consumed 26
producer produced 25
producer produced 9
consumer consumed 9
consumer consumed 25
producer produced 2
consumer consumed 2
producer produced 16
consumer consumed 16
producer produced 3
consumer consumed 3
producer produced 13
consumer consumed 13
producer produced 15
consumer consumed 15
producer produced 11
consumer consumed 11
producer produced 12
consumer consumed 12
liyang@liyang-OptiPlex-7040:~/Desktop/project4$
```

### 4. Conclusion

In this experiment, I understand how to keep thread in sync. Semaphores are useful tool to solve this problem. We can

lock a semaphore when we are doing operations to a buffer if we don't want to let other threads do operations simultaneously.

This experiment let me understand more about the method of thread synchronization. It really improve my skill of multithread programming.