操作系统实验报告

实验三 生产者消费者问题

2019-3-23

北京理工大学 计算机学院

谭超 1120161874

操作系统课程设计实验报告

实验名称：生产者消费者问题

姓名/学号：谭超 1120161874

一、 实验目的

通过模拟消费者生产者问题来对系统的共享机制有一个更为深入的了解与认知。

二、 实验内容

分别在 Windows 系统和 Linux 系统下实现以下问题：

一个大小为 3 的缓冲区，初始为空

- 2 个生产者
  - 随机等待一段时间，往缓冲区添加数据，
  - 若缓冲区已满，等待消费者取走数据后再添加
  - 重复 6 次
- 3 个消费者
  - 随机等待一段时间，从缓冲区读取数据
  - 若缓冲区为空，等待生产者添加数据后再读取
  - 重复 4 次

说明：

- 显示每次添加和读取数据的时间及缓冲区里的数据
- 生产者和消费者用进程模拟

三、 实验环境

Windows:

Windows10 1803

处理器 Intel Core i5-6300HQ 2.3GHz ,RAM 8G

Linux:

VMware Workstation 14.1.3

Ubuntu 4.13.0.36

四、 程序设计与实现

Windows：

在 Windows 系统中通过文件映射的方式实现缓冲区，故而首先要自己定义缓冲区结构、共享内存结构。

```
typedef struct mybuffer
{
    int buff[BUFFER_LENGTH];
    int head;
    int tail;
    int is_empty;
}BUFFER;
```

```
typedef struct sharemem
{
    struct mybuffer data;
    HANDLE sem_empty;
    HANDLE sem_full;
    HANDLE sem_mutex;
}SHM;
```

对于每个进程，我们给其一个 ID，在 main 函数中，我们通过当前进程的 ID 来判断是主进程还是生产者进程或者消费者进程，然后分别执行不同的操作。

```
//对于主控进程
if (nClone == ID_FATHER)
{…
}
//对于生产者进程
else if (nClone >= ID_PRDUCER_START && nClone <= ID_PRODUCER_END)
{…
}
//对于消费者进程
else if (nClone >= ID_CUSTOMER_START && nClone <= ID_CUSTOMER_END)
{…
}
else ;
```

在主进程中，首先要创建一个共享内存区的文件映射，并对共享内存区初始化，因为我的共享内存区内包括了缓冲区和信号量句柄，故而此处的初始化包括缓冲区的初始化和信号量的创建。

```
//初始化缓冲区
struct sharemem * shm = (struct sharemem *)(pFile);
shm->data.head = 0;
shm->data.tail = 0;
shm->data.is_empty=1;
//创建信号量sem_empty、sem_full、sem_mutex并分别初始化为3，0，1
shm->sem_empty = CreateSemaphore(NULL, BUFFER_LENGTH, BUFFER_LENGTH, (LPCSTR)"SEM_EMPTY");
shm->sem_full = CreateSemaphore(NULL, 0, BUFFER_LENGTH, (LPCSTR)"SEM_FULL");
shm->sem_mutex=CreateSemaphore(NULL,1,1,(LPCSTR)"SEM_MUTEX");
```

完成共享内存区的初始化后，就创建生产者进程和消费者进程，然后等待生产者和消费者进程执行完毕，关闭共享内存区。

对于生产者进程，首先通过 OpenFileMapping()打开共享内存区，用 MapViewOfFile()将其映射到当前进程的地址空间。

```
HANDLE hFileMapping = OpenFileMapping(FILE_MAP_ALL_ACCESS, FALSE, SHM_NAME);
LPVOID pFile = MapViewOfFile(hFileMapping, FILE_MAP_ALL_ACCESS, 0, 0, 0);
```

然后通过 OpenSemaphore()打开信号量。

```
HANDLE empty=OpenSemaphore(SEMAPHORE_ALL_ACCESS,FALSE,(LPCSTR)"SEM_EMPTY");
HANDLE full = OpenSemaphore(SEMAPHORE_ALL_ACCESS,FALSE,(LPCSTR)"SEM_FULL");
HANDLE mutex=OpenSemaphore(SEMAPHORE_ALL_ACCESS,FALSE,(LPCSTR)"SEM_MUTEX");
```

在每一次写操作之前先随机睡眠一段时间，然后通过 WaitForSingleObject()函数申请缓冲区空的信号量 empty，再申请对缓冲区互斥操作的信号量 mutex

```
WaitForSingleObject(empty, INFINITE);
WaitForSingleObject(mutex, INFINITE);
```

然后对缓冲区进行写操作，并输出当前时间和缓冲区内的数据，完成之后通过 ReleaseSemaphore()释放缓冲区为满的信号量 full 和互斥操作信号量 mutex。

```
ReleaseSemaphore(full, 1, NULL);
ReleaseSemaphore(mutex, 1, NULL);
```

在该生产者进程完成所有的写操作后，解除共享内存的映射。消费者进程与生产者进程类似，但消费者进程是申请 full 和 mutex 信号量，释放 empty 和 mutex 信号量。

实验结果：

```
PS D:\SourceCode\C++SourceCode\VScode\os_experiment> .\producerCustomer.exe
Main process start.
Current time:  11:03:23
producer 1 write 619 to buffer.
Data in buffer: 619

Current:  11:03:23:
Customer 2 read 619 from buffer.
No data in buffer.

Current time:  11:03:23
producer 2 write 778 to buffer.
Data in buffer: 778
```

```
Current:  11:03:26:
Customer 3 read 788 from buffer.
No data in buffer.

Current time:  11:03:27
producer 2 write 791 to buffer.
Data in buffer: 791

Current:  11:03:27:
Customer 1 read 791 from buffer.
No data in buffer.

Main process ends.
PS D:\SourceCode\C++SourceCode\VScode\os_experiment>
```

Linux：

首先通过 semget()函数创建一个信号量集，其中包含三个信号量，分别是 SEM_EMPTY 表示缓冲区中的空缓冲、SEM_FULL 表示缓冲区中的满缓冲、SEM_MUTEX 表示对缓冲区的互斥使用。然后通过 semctl()初始化三个信号量分别为 3、0、1，在初始化的时候要用到 union semun 结构，这本来应该是已有的结构只要调用即可，但我的 sem.h 头文件中没有这个结构，故而我自己定义了一个：

```
union semun
{
    int val;
    struct semd_ds* buf;
    unsigned short int* array;
    struct seminfo* _buf;
};
```

创建信号量集成功之后便是利用 shmget()创建共享内存区，在此之前，我们要先定义一个缓冲区结构 BUFFER，这样在创建共享内存区的时候就能申请一个缓冲区大小的共享内存区。缓冲区结构如下：

```
typedef struct mybuffer
{
    int buff[BUF_LENGTH];
    int head;                //
    int tail;
    int is_empty;
}BUFFER;
```

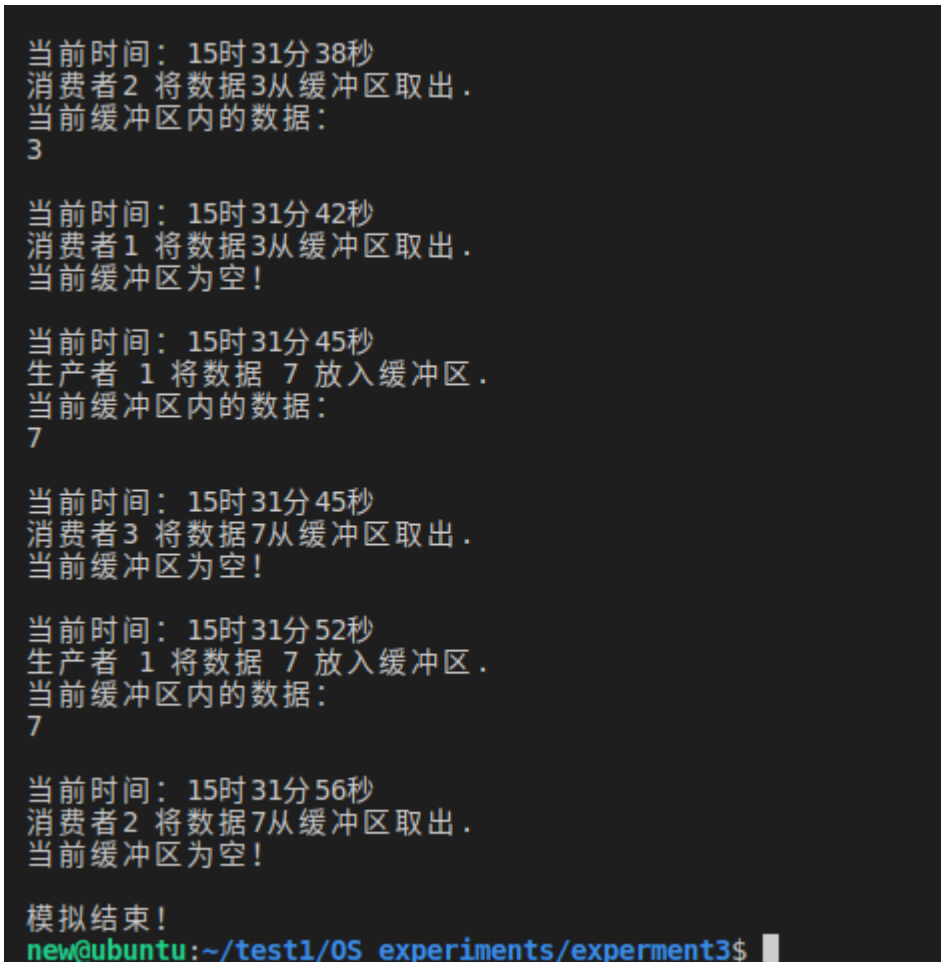
创建共享内存区之后，我们需要利用 shmat()函数将其连接到父进程的地址空间上，然后初始化缓冲区：

```
shmaddr->head = 0;
shmaddr->tail = 0;
shmaddr->is_empty = 1;
```

至此，信号量和缓冲区的创建已经完成，接下来就是创建生产者和消费者进程。对于每个生产者，首先利用 shmat()函数将共享内存区连接到该生产者的地址空间上，然后 sleep(随机时间)，申请一个空缓冲区，如果申请到空缓冲区，则再申请互斥访问权限 SEM_MUTEX，然后向缓冲区中写入一个随机数。并输出当前时间和缓冲区内的数据。操作完成后，释放空缓冲区和互斥信号量。消费者进程与生产者进程类似，只是写操作变为读操作。需要注意的是：缓冲区为空或为满时的标志都是 head==tail,此时我门需要根据所在的进程进行判断，因为生产者进程中缓冲区一定不为空，消费者进程中缓冲区一定不为满。另外也可通过 BUFFER 结构中的 is_empty 来判断缓冲区是空还是满。最后，父进程要等待所有生产者和消费者进程运行结束之后，删除信号量集和共享内存

实验结果：

```
new@ubuntu:~/test1/OS_experiments/experment3$ gcc -o producerConsumer producerConsumer.c
new@ubuntu:~/test1/OS_experiments/experment3$ ./producerConsumer
当前时间：15时31分12秒
生产者 2 将数据 3 放入缓冲区.
当前缓冲区内的数据：
3

当前时间：15时31分13秒
生产者 2 将数据 1 放入缓冲区.
当前缓冲区内的数据：
3 1

当前时间：15时31分13秒
生产者 1 将数据 4 放入缓冲区.
当前缓冲区内的数据：


当前时间：15时31分13秒
消费者1 将数据3从缓冲区取出.
当前缓冲区内的数据：
1 4

当前时间：15时31分19秒
消费者3 将数据1从缓冲区取出.
当前缓冲区内的数据：
4

当前时间：15时31分20秒
```



五、 实验收获与体会

在本次实验中，我对 Windows 和 Linux 系统中的信号量、共享内存区相关的 API 的使用更加熟悉。同时也对信号量的作用与意义有了更加深入的理解。对于进程的并行执行更加印象深刻。在 Windows 的编程中，在生产者和消费者进程中，不能用共享内存区的句柄保存打开信号量返回的句柄，而要在各个进程中重新申请句柄来保存，否则可能会出现生产者向已满的缓冲区写数据、消费者从空的缓冲区内读数据等情况。