计算机学院_操作系统_课程实验报告

实验题目： 进程同步		学号：202200130048
日期：10.30	班级： 6	姓名： 陈静雯
Email：1205037094@qq.com		

实验步骤与现象：

1. 示例实验

```
orange@orange-VirtualBox:~/czsystem/test4$ ./producer 1
2856 producer put: A to Buffer[0]
2856 producer put: B to Buffer[1]
2856 producer put: C to Buffer[2]
2856 producer put: D to Buffer[3]
2856 producer put: E to Buffer[4]
2856 producer put: F to Buffer[5]
2856 producer put: G to Buffer[6]
2856 producer put: H to Buffer[7]
```

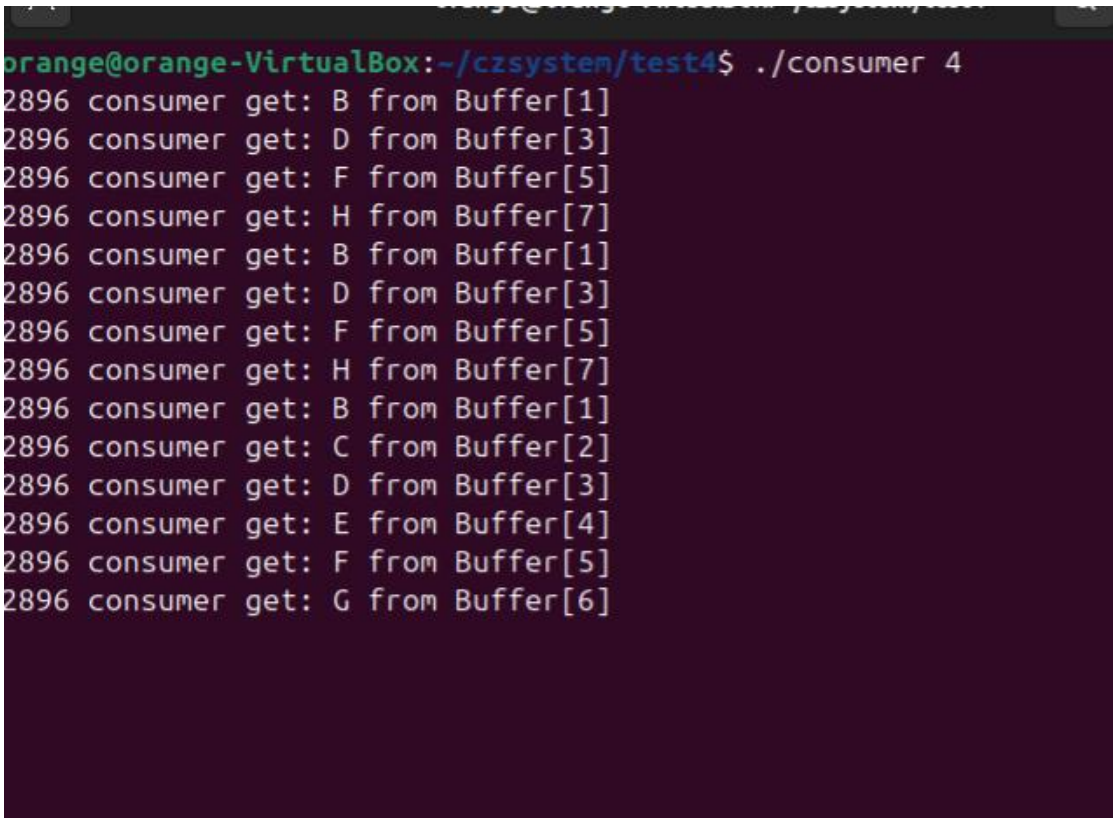
2856 进程在向共享内存中连续写入了 8 个字符后因为缓冲区满而阻塞。

```
orange@orange-VirtualBox: ~/czsystem/test4
orange@orange-VirtualBox:~/czsystem/test4$ ./producer 3
```

该生产者进程因为缓冲区已满而立即阻塞。

```
orange@orange-VirtualBox:~/czsystem/test4$ ./consumer 2
2888 consumer get: A from Buffer[0]
2888 consumer get: B from Buffer[1]
2888 consumer get: C from Buffer[2]
2888 consumer get: D from Buffer[3]
2888 consumer get: E from Buffer[4]
2888 consumer get: F from Buffer[5]
2888 consumer get: G from Buffer[6]
2888 consumer get: H from Buffer[7]
2888 consumer get: A from Buffer[0]
2888 consumer get: C from Buffer[2]
2888 consumer get: E from Buffer[4]
2888 consumer get: G from Buffer[6]
2888 consumer get: A from Buffer[0]
2888 consumer get: C from Buffer[2]
2888 consumer get: E from Buffer[4]
2888 consumer get: G from Buffer[6]
2888 consumer get: A from Buffer[0]
```


orange@orange-VirtualBox:~/czsystem/test4$ ./producer 1
2856 producer put: A to Buffer[0]
2856 producer put: B to Buffer[1]
2856 producer put: C to Buffer[2]
2856 producer put: D to Buffer[3]
2856 producer put: E to Buffer[4]
2856 producer put: F to Buffer[5]
2856 producer put: G to Buffer[6]
2856 producer put: H to Buffer[7]
^[
2856 producer put: A to Buffer[0]
2856 producer put: C to Buffer[2]
2856 producer put: E to Buffer[4]
2856 producer put: G to Buffer[6]
2856 producer put: A to Buffer[0]
2856 producer put: C to Buffer[2]
2856 producer put: E to Buffer[4]
2856 producer put: G to Buffer[6]
2856 producer put: A to Buffer[0]
^C


orange@orange-VirtualBox:~/czsystem/test4$ ./consumer 4
2896 consumer get: B from Buffer[1]
2896 consumer get: D from Buffer[3]
2896 consumer get: F from Buffer[5]
2896 consumer get: H from Buffer[7]
2896 consumer get: B from Buffer[1]
2896 consumer get: D from Buffer[3]
2896 consumer get: F from Buffer[5]
2896 consumer get: H from Buffer[7]
2896 consumer get: B from Buffer[1]
2896 consumer get: C from Buffer[2]
2896 consumer get: D from Buffer[3]
2896 consumer get: E from Buffer[4]
2896 consumer get: F from Buffer[5]
2896 consumer get: G from Buffer[6]

```
orange@orange-VirtualBox:~/czsystem/test4$ ./producer 3
2877 producer put: B to Buffer[1]
2877 producer put: D to Buffer[3]
2877 producer put: F to Buffer[5]
2877 producer put: H to Buffer[7]
2877 producer put: B to Buffer[1]
2877 producer put: D to Buffer[3]
2877 producer put: F to Buffer[5]
2877 producer put: H to Buffer[7]
2877 producer put: B to Buffer[1]
2877 producer put: C to Buffer[2]
2877 producer put: D to Buffer[3]
2877 producer put: E to Buffer[4]
2877 producer put: F to Buffer[5]
2877 producer put: G to Buffer[6]
2877 producer put: H to Buffer[7]
^C
```

由于消费者进程读出了写入缓冲区的字符，生产者重新被唤醒继续向读过的缓冲区单元中同步的写入字符。

用 Ctrl+c 将两生产者进程打断，两消费者进程在读空缓冲区后而阻塞。反之，用 Ctrl+c 将两消费者进程打断，两生产者进程在写满缓冲区后而阻塞。

2. 独立实验

```
orange@orange-VirtualBox:~/czsystem/test4/tobaaco$ ./producer
第2758号进程放入胶水和烟草:A 到 Buffer[0]
第2758号进程放入胶水和纸:B 到 Buffer[1]
第2758号进程放入纸和烟草:C 到 Buffer[2]
第2758号进程放入胶水和烟草:D 到 Buffer[3]
```

缓冲区大小为 4，生产者阻塞

消费者开始消费两种物品后，生产者继续工作

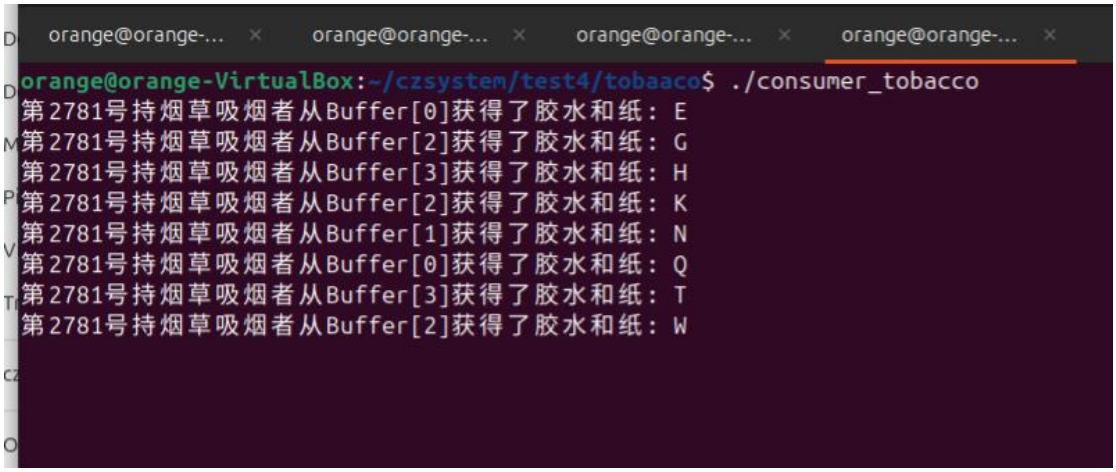
```
orange@orange-VirtualBox:~/czsystem/test4/tobaaco$ ./producer
第2758号进程放入胶水和烟草:A 到 Buffer[0]
第2758号进程放入胶水和纸:B 到 Buffer[1]
第2758号进程放入纸和烟草:C 到 Buffer[2]
第2758号进程放入胶水和烟草:D 到 Buffer[3]
第2758号进程放入胶水和纸:E 到 Buffer[0]
第2758号进程放入纸和烟草:F 到 Buffer[1]
第2758号进程放入胶水和烟草:G 到 Buffer[2]
第2758号进程放入胶水和纸:H 到 Buffer[3]
第2758号进程放入纸和烟草:I 到 Buffer[0]
第2758号进程放入胶水和烟草:J 到 Buffer[1]
第2758号进程放入胶水和纸:K 到 Buffer[2]
第2758号进程放入纸和烟草:L 到 Buffer[3]
第2758号进程放入胶水和烟草:M 到 Buffer[0]
第2758号进程放入胶水和纸:N 到 Buffer[1]
第2758号进程放入纸和烟草:O 到 Buffer[2]
第2758号进程放入胶水和烟草:P 到 Buffer[3]
第2758号进程放入胶水和纸:Q 到 Buffer[0]
第2758号进程放入纸和烟草:R 到 Buffer[1]
```

```
orange@orange-VirtualBox:~/czsystem/test4/tobaaco$ ./consumer_glue
第2767号持纸胶水吸烟者从Buffer[0]获得了纸和烟草: A
第2767号持纸胶水吸烟者从Buffer[3]获得了纸和烟草: D
第2767号持纸胶水吸烟者从Buffer[0]获得了纸和烟草: I
第2767号持纸胶水吸烟者从Buffer[3]获得了纸和烟草: L
第2767号持纸胶水吸烟者从Buffer[2]获得了纸和烟草: O
第2767号持纸胶水吸烟者从Buffer[1]获得了纸和烟草: R
第2767号持纸胶水吸烟者从Buffer[0]获得了纸和烟草: U
```

拥有胶水的消费纸和烟草

```
orange@orange-...  orange@orange-...  orange@orange-...  orange@orange-...
orange@orange-VirtualBox:~/czsystem/test4/tobaaco$ ./consumer_paper
第2774号持纸吸烟者从Buffer[1]获得了胶水和烟草: B
第2774号持纸吸烟者从Buffer[2]获得了胶水和烟草: C
第2774号持纸吸烟者从Buffer[1]获得了胶水和烟草: F
第2774号持纸吸烟者从Buffer[1]获得了胶水和烟草: J
第2774号持纸吸烟者从Buffer[0]获得了胶水和烟草: M
第2774号持纸吸烟者从Buffer[3]获得了胶水和烟草: P
第2774号持纸吸烟者从Buffer[2]获得了胶水和烟草: S
第2774号持纸吸烟者从Buffer[1]获得了胶水和烟草: V
```

拥有纸的消费胶水和烟草



拥有烟草的消费纸和胶水
生产队列为空，消费者阻塞

结论分析：

1. 真实操作系统中提供的并发进程同步机制是怎样实现和解决同步问题的

（1）互斥锁（Mutex）：用于保护共享资源，确保同一时间只有一个进程可以访问该资源。

（2）信号量（Semaphore）：用于控制对共享资源的访问，可以表示资源的数量或状态。操作系统内核提供了 P（down）和 V（up）操作来操作信号量。

（3）条件变量（Condition Variable）：用于进程间的条件等待和通知。结合互斥锁使用，进程可以在某个条件不满足时等待，直到其他进程通知条件满足。

2. 它们是怎样应用操作系统教材中讲解的进程同步原理的？

（1）互斥锁的应用：

在生产者/消费者问题中，互斥锁 mutex 用于保护对共享缓冲区的访问，确保同一时间只有一个进程可以修改缓冲区内容。

```
//如果无产品消费者阻塞
down(cons_sem);
//如果另一消费者正在取产品，本消费者阻塞
down(cmtx_sem);
//用读一字符的形式模拟消费者取产品，报告本进程号和获取的字符及读取的位置
sleep(rate);
printf("%d consumer get: %c from Buffer[%d]\n",getpid(),buff_ptr[*cget_ptr],*cg
```

在抽烟者问题中，互斥锁 mutex 用于保护对桌子上材料的访问，确保同一时间只有一个抽烟者可以取走材料。

（2）信号量的应用：

在生产者/消费者问题中，信号量 full 表示缓冲区的空闲槽的数量，通过 down 和 up 操作来控制生产者和消费者的同步。

```
//信号量使用的变量
prod_key = 201;//生产者同步信号量键值
pmtx_key = 202;//生产者互斥信号量键值
cons_key = 301;//消费者同步信号量键值
cmtx_key = 302;//消费者互斥信号量键值
sem_flg = IPC_CREAT | 0644;
//生产者同步信号量初值设为缓冲区最大可用量
sem_val = buff_num;
```

在抽烟者问题中，信号量 GlueTobacco、GluePaper 和 PaperTobacco 分别表示桌子上是否有对应的材料组合，通过 down 和 up 操作来唤醒对应的抽烟者。

```
buff_key = 101; // 缓冲区任给的键值
buff_num = 4; // 缓冲区任给的长度
pput_key = 102; // 生产者放产品指针的键值
pput_num = 1; // 指针数
shm_flg = IPC_CREAT | 0644; // 共享内存读写权限
// 获取缓冲区使用的共享内存,buff_ptr 指向缓冲区首地址
buff_ptr = (char*)set_shm(buff_key, buff_num, shm_flg);
// 获取生产者放产品位置指针 pput_ptr
pput_ptr = (int*)set_shm(pput_key, pput_num, shm_flg);
// 信号量使用的变量
full_key = 201;   //同步
mutex_key = 202;  //互斥
GlueTobacco_key = 203;
GluePaper_key = 204;
PaperTobacco_key = 205;
sem_flg = IPC_CREAT | 0644;
```

3. 对应教材中信号量的定义，说明信号量机制是怎样完成进程的互斥和同步的？

（1）信号量的定义

信号量（Semaphore）是一种用于解决多进程之间同步和互斥问题的数据结构。它是一个非负整数，表示资源的数量或状态。信号量支持两个基本操作：P（也称为 wait 或 down）和 V（也称为 signal 或 up）。

P 操作（down）：尝试获取资源。如果信号量的值大于 0，则将信号量的值减 1，并继续执行；如果信号量的值为 0，则当前进程会被阻塞，直到信号量的值大于 0。

```c
int down(int sem_id) //P 操作
{
    struct sembuf buf;
    buf.sem_op = -1;
    buf.sem_num = 0;
    buf.sem_flg = SEM_UNDO;
    if((semop(sem_id,&buf,1)) <0)
    {
        perror("down error ");
        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS;
}
```

V 操作（up）：释放资源。将信号量的值加 1，并唤醒一个因执行 P 操作而被阻塞的进程

```c
int up(int sem_id) //V 操作
{
    struct sembuf buf;
    buf.sem_op = 1;
    buf.sem_num = 0;
    buf.sem_flg = SEM_UNDO;
    if((semop(sem_id,&buf,1)) <0)
    {
        perror("up error ");
        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS;
}
```

（2）互斥：

互斥锁（Mutex）：通常使用一个初始值为 1 的信号量来实现互斥锁。这个信号量表示一个共享资源是否可用。

工作原理：当一个进程需要访问共享资源时，它执行 P 操作。如果信号量的值为 1，表示资源可用，信号量的值减 1，进程继续执行。如果信号量的值为 0，表示资源已被其他进程占用，当前进程会被阻塞，直到资源可用。当进程完成对共享资源的操作后，它执行 V 操作，将信号量的值加 1，表示资源已释放，并唤醒一个因执行 P 操作而被阻塞的进程。

（3）同步：

同步信号量：用于协调多个进程之间的操作顺序。通常使用一个初始值为 0 的信号量来表示某个事件是否已经发生。

工作原理：当一个进程完成某项任务后，它执行 V 操作，将信号量的值加 1，表示任务已完成。其他依赖该任务完成的进程执行 P 操作。如果信号量的值为 0，表示任务尚未完成，当前进程会被阻塞，直到任务完成。一旦任务完成，信号量的值变为 1，被阻塞的进程会被唤醒，继续执行

4. 其中信号量的初值和其值的变化的物理意义是什么？

（1）信号量的初值：

互斥信号量：初始值通常为 1，表示共享资源最初是可用的。

同步信号量：初始值通常为 0，表示某个事件尚未发生或某个任务尚未完成。

（2）信号量值的变化：

P 操作（down）：

互斥信号量：将信号量的值减 1，表示占用了一个资源。如果信号量的值变为 0，表示资源已被占用；如果信号量的值小于 0，表示资源已被占用且有进程在等待。

同步信号量：将信号量的值减 1，表示等待某个事件的发生。如果信号量的值为 0，表示事件尚未发生，当前进程会被阻塞。

V 操作（up）：

互斥信号量：将信号量的值加 1，表示释放了一个资源。如果有进程在等待资源，其中一个会被唤醒。

同步信号量：将信号量的值加 1，表示某个事件已经发生。如果有进程在等待该事件，其中一个会被唤醒。

5. 使用多于 4 个的生产者和消费者，以各种不同的启动顺序、不同的执行速率检测以上示例程序和独立实验程序也都能满足同步的要求。