exercise1：请回答一下，什么情况下会出现死锁。

答：当每个哲学家都拿起了左边的叉子时，会导致每个哲学家都无法拿到右边的叉子，导致死锁。

exercise2：说一下该方案有什么不足？（答出一点即可）

答：同时只允许一个哲学家吃面条。

exercise3：正确且高效的解法有很多，请你利用信号量 PV 操作设计一种正确且相对高效（比方案 2 高效）的哲学家吃饭算法。（其实网上一堆答案，主要是让大家多看看不同的实现。）

答：每个哲学家取到手边的两把叉子才开始吃面，否则一把都不取

```
#define N 5                 // 哲学家个数

semaphore fork[5];          // 信号量初值为 1

semaphore mutex;            // 互斥信号量，初值 1void philosopher(int i){    // 哲学家编号：
0-4

  while(TRUE){

    think();                // 哲学家在思考

    P(mutex);               // 进入临界区

    P(fork[i]);             // 去拿左边的叉子

    P(fork[(i+1)%N]);       // 去拿右边的叉子

    V(mutex);               // 退出临界区

    eat();                  // 吃面条

    V(fork[i]);             // 放下左边的叉子

    V(fork[(i+1)%N]);       // 放下右边的叉子

      }

}
```

exercise4：为什么要用两个信号量呢？emptyBuffers 和 fullBuffer 分别有什么直观含义？

答：因为需要实现两个条件同步：缓冲区空时，消费者必须等待生产者；缓冲区满时，生产者必须等待消费者。

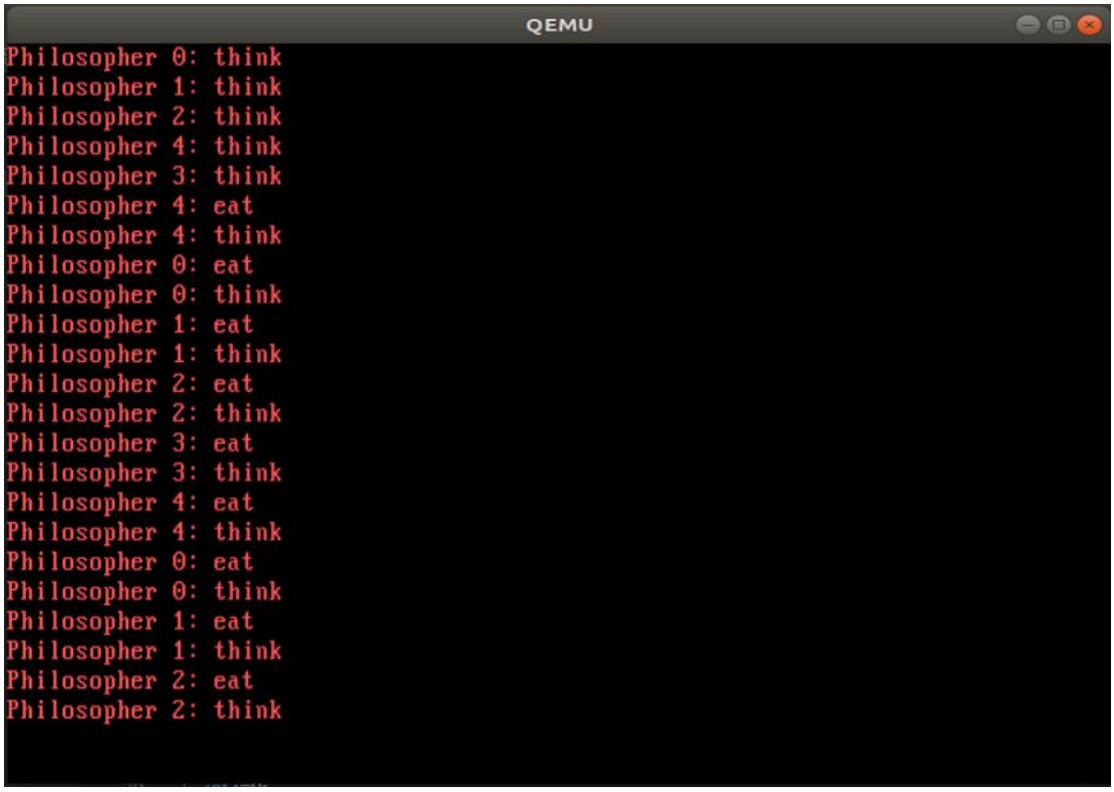
EmptyBuffers 代表缓冲区剩余空间的数量。

FullBuffers 代表缓冲区可取走资源的数量。

task3：完成 app 里面的下列问题，在报告里放上运行截图（注意在写其中一个问题时，把别的代码注释掉）。

哲学家就餐问题

截图：



代码：

```
//哲学家


int j=0;
int ret;
sem_t forks[5];
sem_t mutex;
sem_init(&mutex, 1);
```

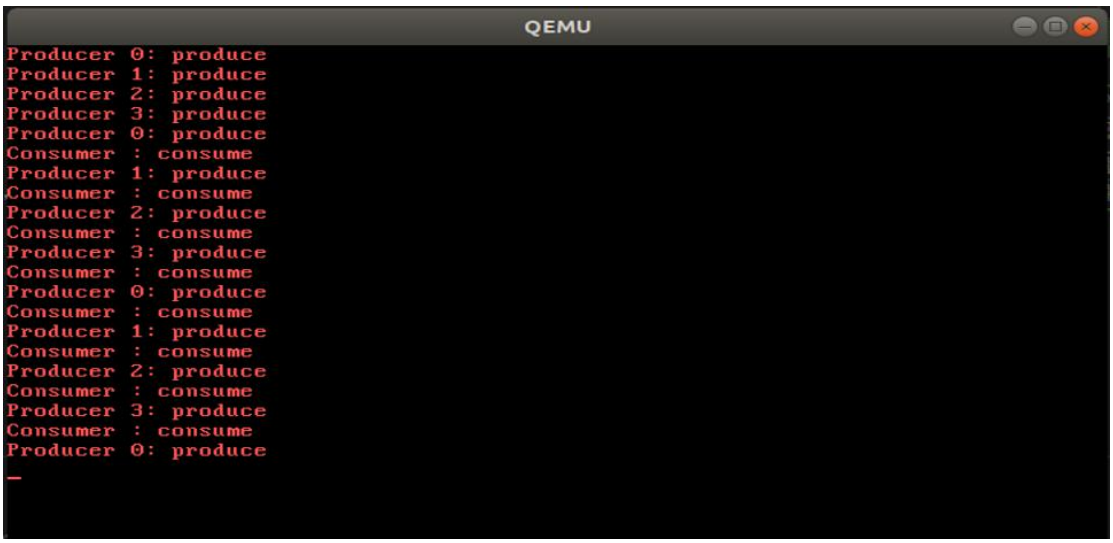
```
for (int i=0;i<5;i++)
sem_init(&forks[i], 1);

for(;j<4;++j)
{
        ret = fork();
        if(ret == 0)
                break;
        else if(ret < 0)
        {
                printf("fork error\n");
                exit(1);
        }
}

while(1)
{
        printf("Philosopher %d: think\n", j);
        sleep(128);
        sem_wait(&mutex);
        sleep(128);                     //
    sem_wait(&forks[j]);
        sleep(128);                     //
    sem_wait(&forks[(j+1)%5]);
        sleep(128);                     //
    sem_post(&mutex);
        sleep(128);                     //
    printf("Philosopher %d: eat\n", j);
        sleep(128);
    sem_post(&forks[j]);
        sleep(128);                     //
    sem_post(&forks[(j+1)%5]);
        sleep(128);                     //
}
```

生产者-消费者问题

截图：



代码：

```
//生产者消费者问题
sem_t mutex;
sem_t empty;
sem_t full;
sem_init(&empty,5);          //可以使用的空缓冲区数
sem_init(&full,0);                  //缓冲区内可以使用的产品数
sem_init(&mutex,1);               //互斥信号量
int in=0;                                                    //放入缓冲区指针
int out=0;                                 //取出缓冲区指针


        int j=0;
        int ret;

        for(;j<4;++j)
        {
                ret = fork();
                if(ret == 0)
                        break;
                else if(ret < 0)
                {
                        printf("fork error\n");
                        exit(1);
                }
        }

        while(1)
        {
                if(j==4)//consumer
                {
```

```
                sem_wait(&full);
                sem_wait(&mutex);
                out=(out+1)%5;
                sem_post(&mutex);
                sem_post(&empty);
                printf("Consumer : consume\n");
                sleep(128);

        }
        else//producer
        {
                sem_wait(&empty);
                sem_wait(&mutex);
                in=(in+1)%5;
                printf("Producer %d: produce\n", j);
                sleep(128);
                sem_post(&mutex);
                sem_post(&full);
        }
}
```