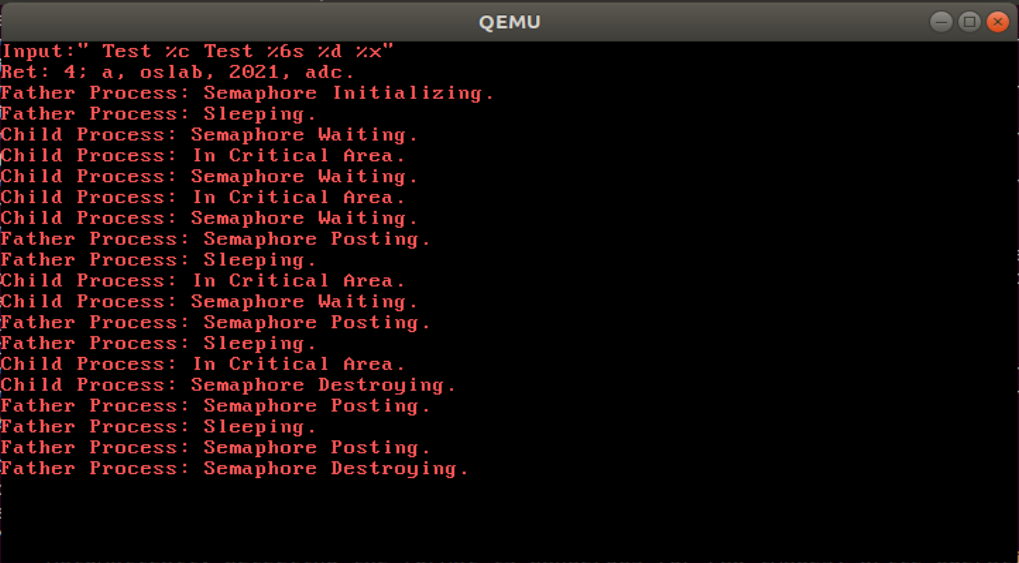
姓名：张涵之 学号：191220154 邮箱：[1683762615@qq.com](mailto:1683762615@qq.com)

实验进度：我完成了3.1-3.3.1的必做内容，选做内容只实现了生产者-消费者

实验结果：下图为3.1-3.2测试代码运行效果



解决进程同步问题（3.3）的结果将在思考和总结部分展示

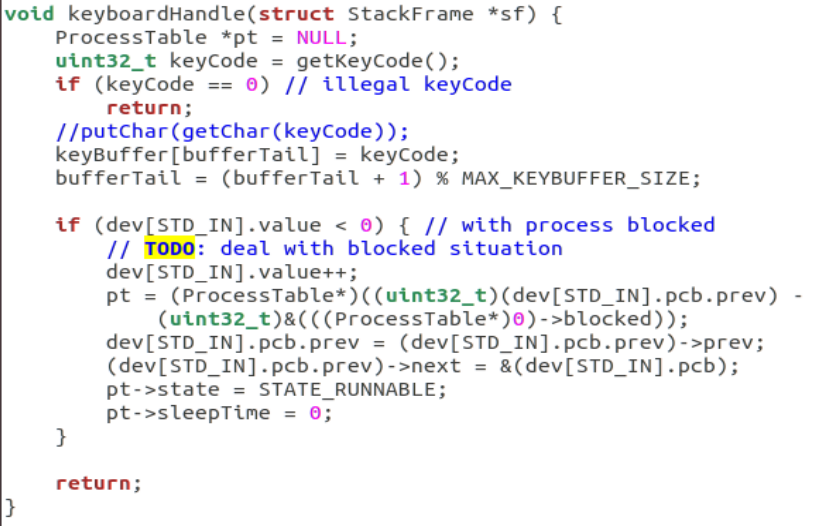
实验修改的代码位置：

3.1. 实现格式化输入函数 lab4/kernel/kernel/irqHandle.c

keyboardHandle：框架代码已经实现了将读取到的keyCode放到keyBuffer中

根据手册中的示例代码，从Device STD\_IN上阻塞的进程列表取出一个进程

将这个进程的state设为RUNNABLE，sleepTime设为0，实现该进程的唤醒



syscallReadStdIn：最多只有一个进程被阻塞在dev[STD\_IN]上，后来的返回-1

即如果dev[STD\_IN].value < 0，证明此时已有多个进程想读，直接设置eax为1

如果dev[STD\_IN].value == 0，则需要将当前进程阻塞在dev[STD\_IN]上

根据手册中的示例代码，可以将current线程加到Device STD\_IN的阻塞列表

将进程的state设为BLOCKED，调用timerHandle唤醒进程，读keyBuffer中的数据

根据手册中的示例代码，可以依次取出字符character传到用户进程

用计数器i统计实际读取的字节数，并通过设置当前进程的eax进行返回



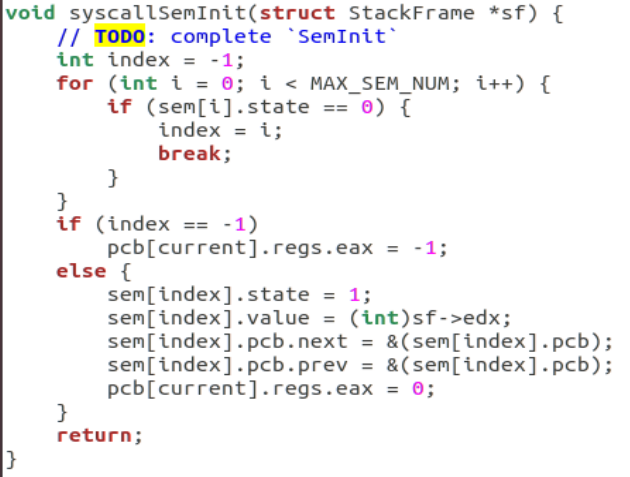
3.2. 实现信号量

syscallSemInit：在sem数组中寻找未用（state==0）的信号量

若找不到则初始化失败，通过eax设置返回值为-1

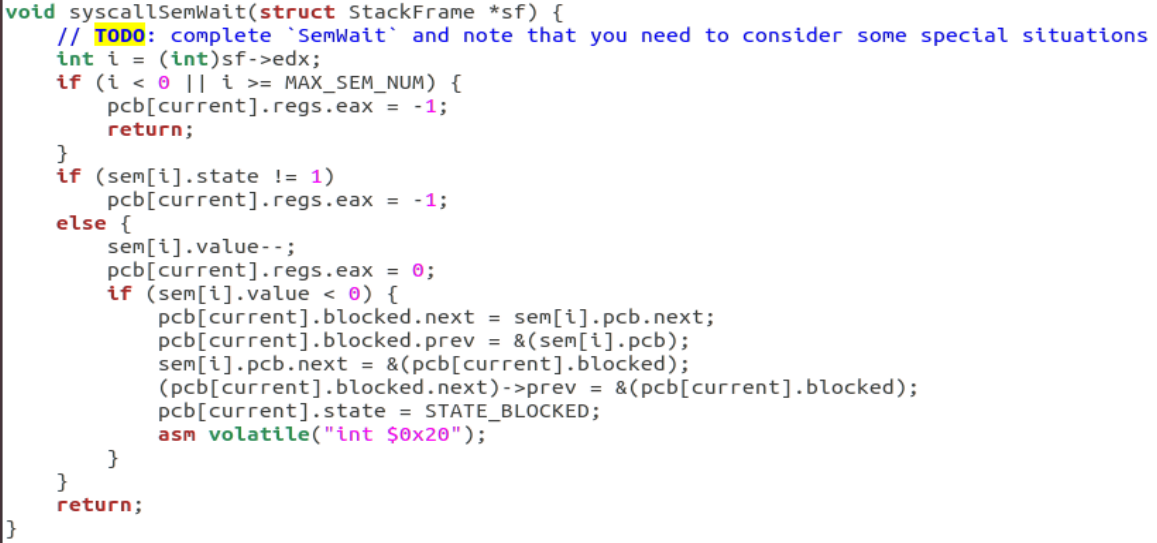
若找到了则初始化成功，将state设为1，value设为指定的初始值

仿照initSem完成信号量的初始化，通过eax设置返回值为0



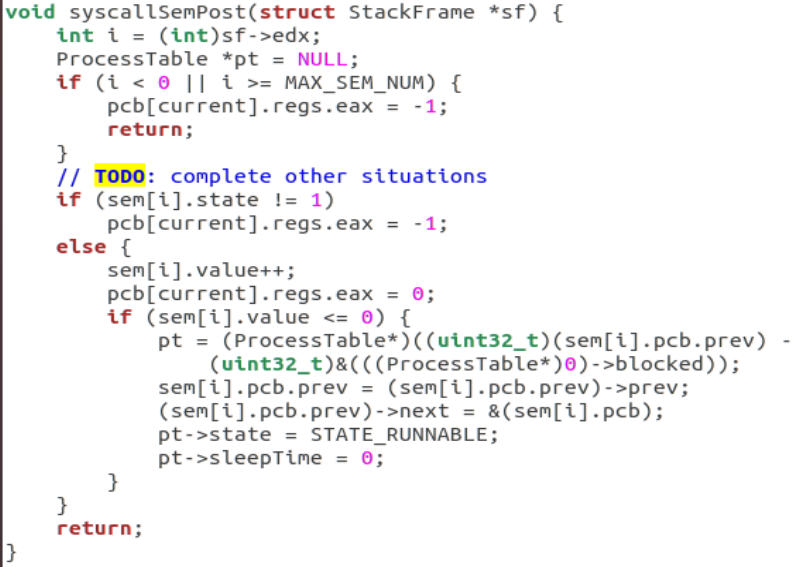
syscall\_wait：若下标i不合法（越界）或信号量未用则操作失败，返回-1

否则使对应sem的value减一，若value取值小于0则阻塞自身，返回0

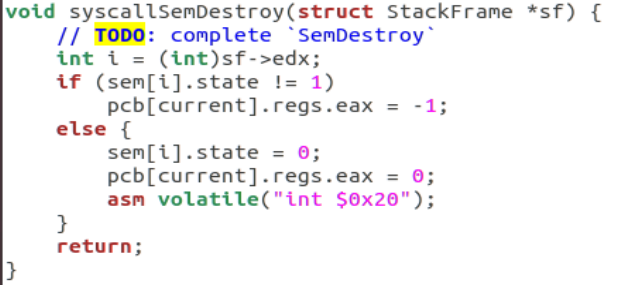


syscall\_post：与syscall\_wait类似，若操作失败则返回值为-1，否则返回值为0

使对应sem的value增一，若value取值不大于0则释放一个阻塞进程

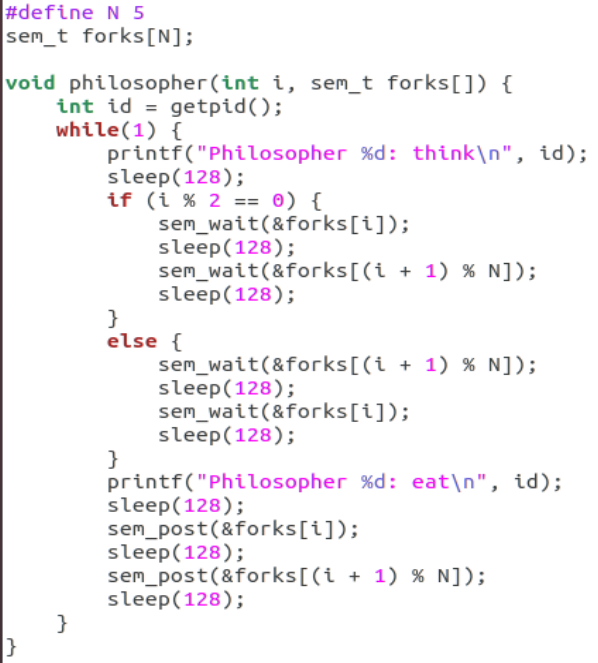


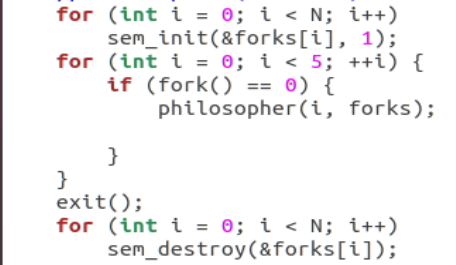
syscallSemDestroy：设置对应state为0即可，失败返回-1，成功返回0



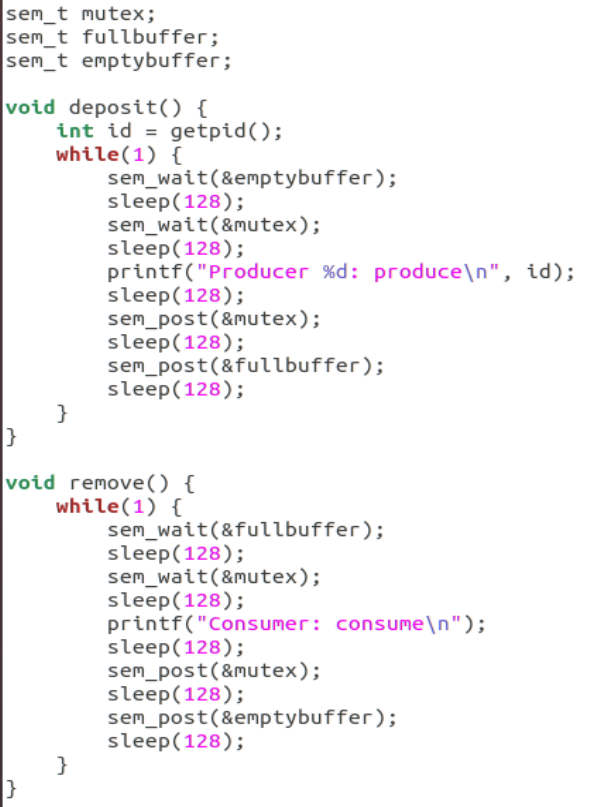
3.3. 解决进程同步问题

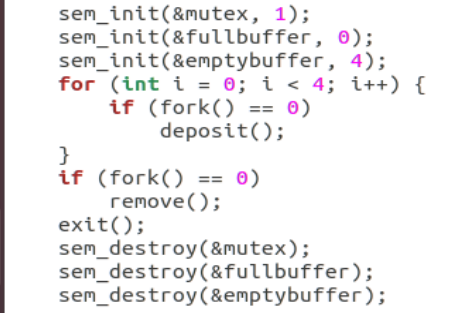
1. 哲学家就餐问题



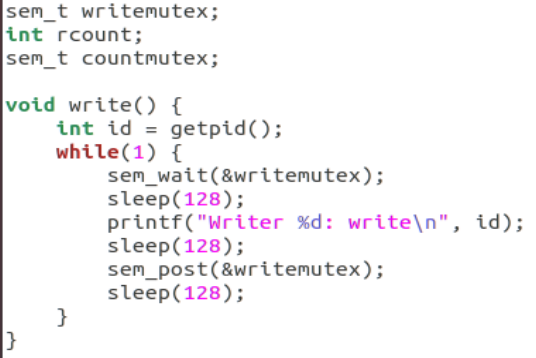


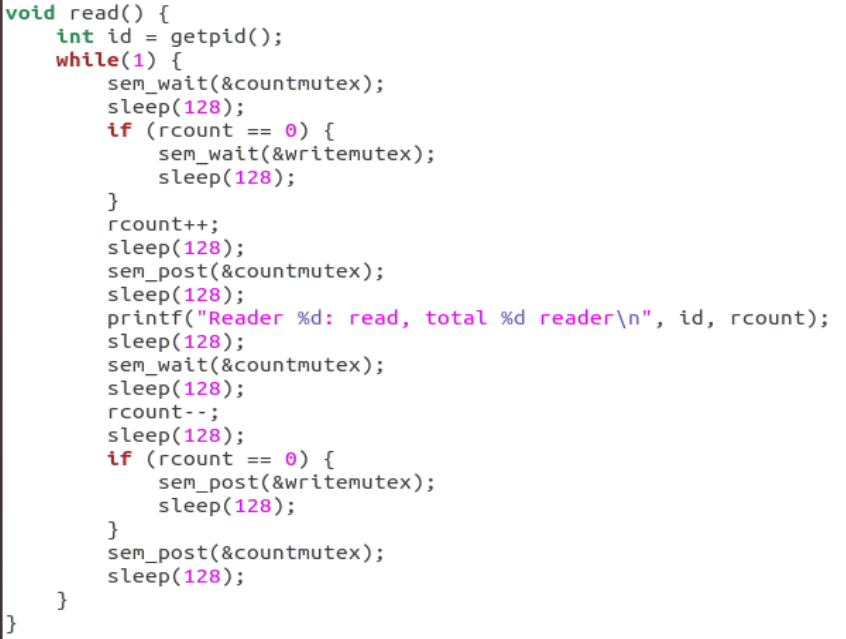
1. 生产者-消费者问题

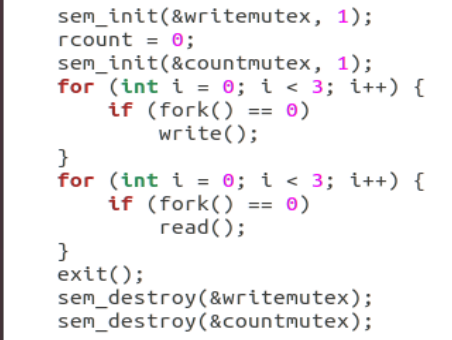




1. 读者-写者问题（失败了）







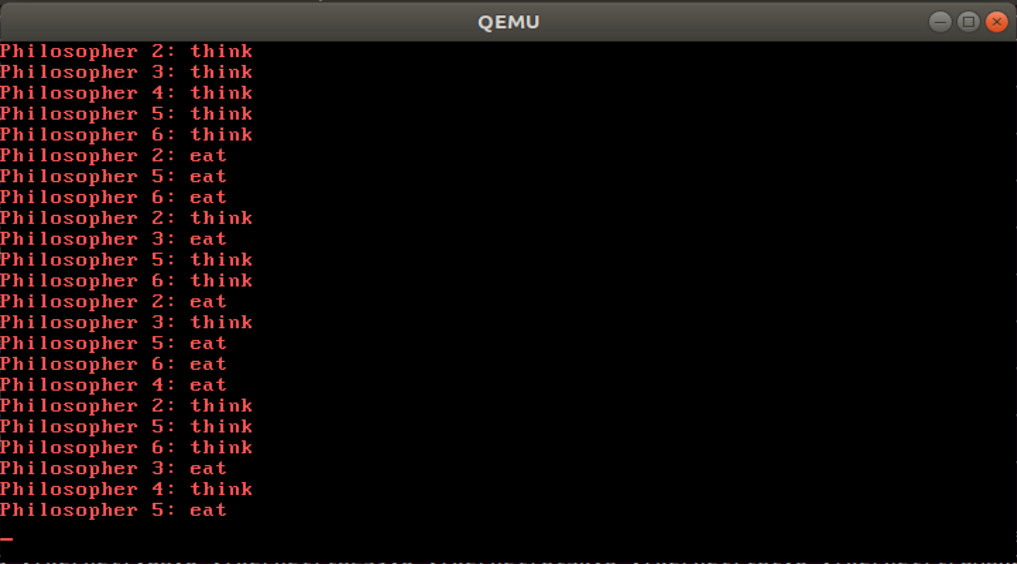
思考和总结：

分析信号量的测试代码，可见其原理大致是子进程申请信号量进入临界区，父进程归还。

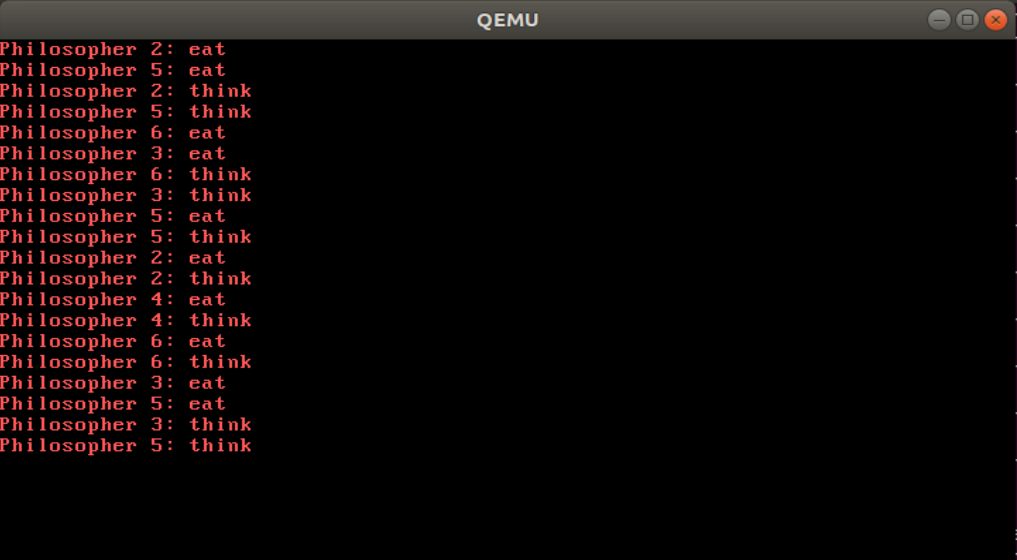
分析哲学家问题的伪代码和输出结果，可以得出哲学家和叉子的序号关系：

哲学家： 2 3 4 5 6

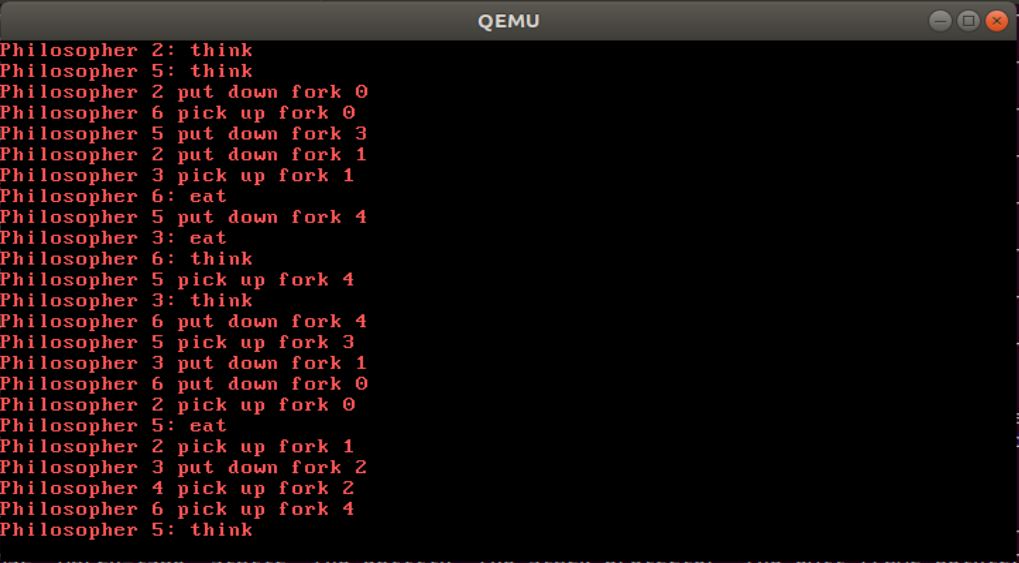
叉子： 0 1 2 3 4 0



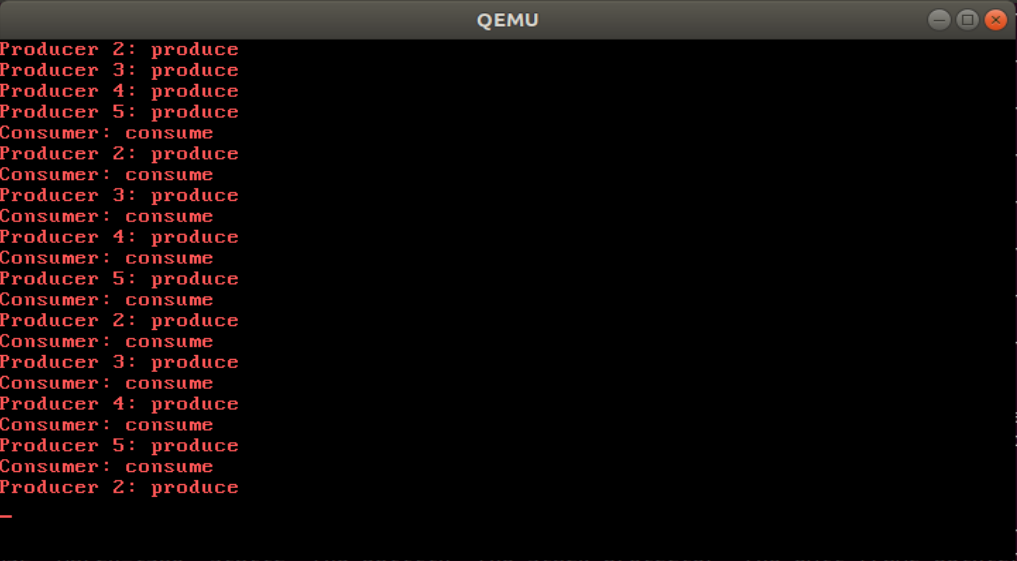
这个输出结果看起来有点奇怪，好像三个以上的哲学家可以同时吃。但5 eat和6 eat之间的时间间隔极长，而6 eat和2 think之间几乎没有时差，由于2和5的think在两个fork都已被释放，再经过一个sleep才输出，推测哲学家2和5已经分别释放了叉子0和4，6开始吃，此时由于叉子1已经释放，叉子2原本就未被占用，所以3也开始吃，然后才输出5 think的信息，我试着把think的输出语句放到sem\_post两个fork前面，效果如下：



可见逻辑基本正常了，每一时刻有且只有两个人同吃，有人放下叉子开始思考，其他人才能开始吃。继续加入调试输出语句，每次执行sem\_wait、sem\_post与sleep之间输出是哪位哲学家拿起或放下了哪个叉子，输出的结果中拿起放下叉子的行为比较符合预期：



生产者-消费者的buffer大小被设为4，可见4名生产者先轮流生产，将buffer填满，在这以后，consumer每次取出一个产品，就有一个生产者放入新的产品：



在读者-写者程序中，注意到读写进程可以互斥，但读的进程不但不能同步，而且会产生死锁现象，输出调试信息发现全局变量Rcount无法被各子进程共享，即在每个进程中都是独立的，不能同步修改。怎么会这样呢？查阅资料发现fork操作和pthread create线程工作原理不同，fork得到的子进程拥有独立的代码段和数据段，即使不同进程中全局变量的逻辑地址是相同的，它们实际在内存空间中的物理地址是不同的。要想在fork出的父子进程中正确更新Rcount需要共享内存、管道通信等方法，由于时间和技术有限这里不作实现。