Lab4 进程同步

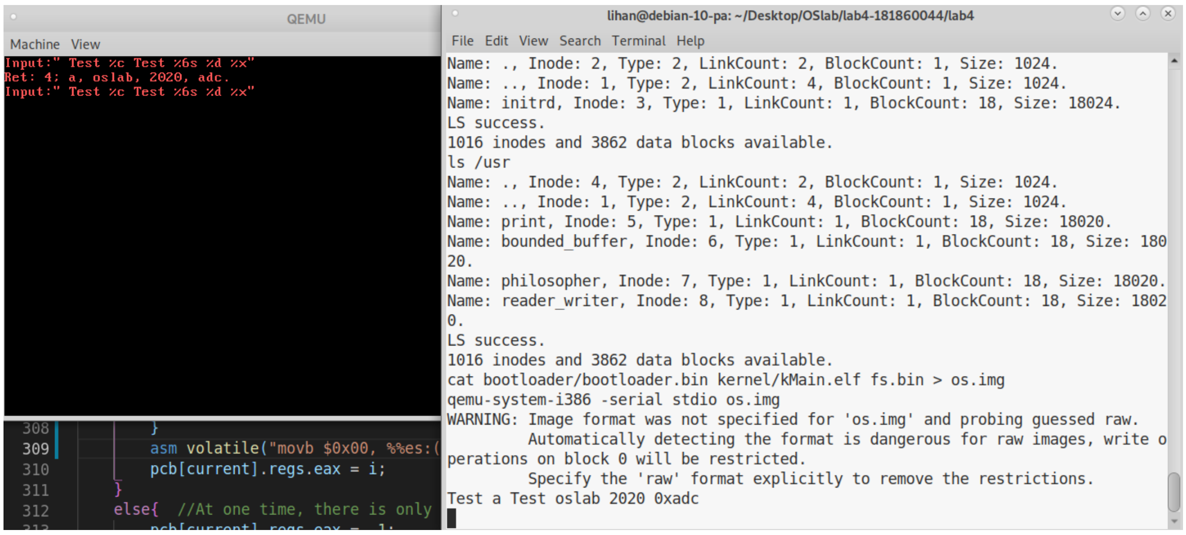
181860044 李翰

邮箱： 181860044@smail.nju.edu.cn

1. 实验进度：完成了所有内容（包括选做的随机函数）

2. 实验结果：

2.1 实现格式化输入函数



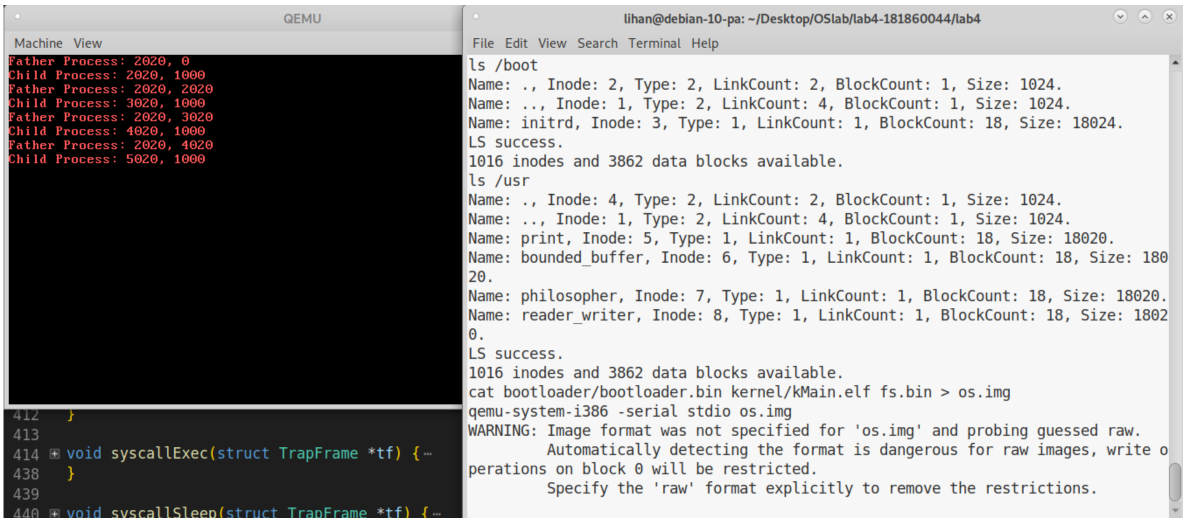
```
Input:" Test %c Test %6s %d %x"
Ret: 4; a, oslab, 2020, adc.
Input:" Test %c Test %6s %d %x"
```

```
        }
        asm volatile("movb $0x00, %%es:(
        pcb[current].regs.eax = i;
    }
    else{  //At one time, there is only
```

lihan@debian-10-pa: ~/Desktop/OSlab/lab4-181860044/lab4

```
Name: ., Inode: 2, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: initrd, Inode: 3, Type: 1, LinkCount: 1, BlockCount: 18, Size: 18024.
LS success.
1016 inodes and 3862 data blocks available.
ls /usr
Name: ., Inode: 4, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: print, Inode: 5, Type: 1, LinkCount: 1, BlockCount: 18, Size: 18020.
Name: bounded_buffer, Inode: 6, Type: 1, LinkCount: 1, BlockCount: 18, Size: 180
20.
Name: philosopher, Inode: 7, Type: 1, LinkCount: 1, BlockCount: 18, Size: 18020.
Name: reader_writer, Inode: 8, Type: 1, LinkCount: 1, BlockCount: 18, Size: 1802
0.
LS success.
1016 inodes and 3862 data blocks available.
cat bootloader/bootloader.bin kernel/kMain.elf fs.bin > os.img
qemu-system-i386 -serial stdio os.img
WARNING: Image format was not specified for 'os.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write o
perations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
Test a Test oslab 2020 0xadc
```

2.2 实现进程通信


QEMU
Machine View
lihan@debian-10-pa: ~/Desktop/OSlab/lab4-181860044/lab4
File Edit View Search Terminal Help

```
Father Process: 2020, 0
Child Process: 2020, 1000
Father Process: 2020, 2020
Child Process: 3020, 1000
Father Process: 2020, 3020
Child Process: 4020, 1000
Father Process: 2020, 4020
Child Process: 5020, 1000
```

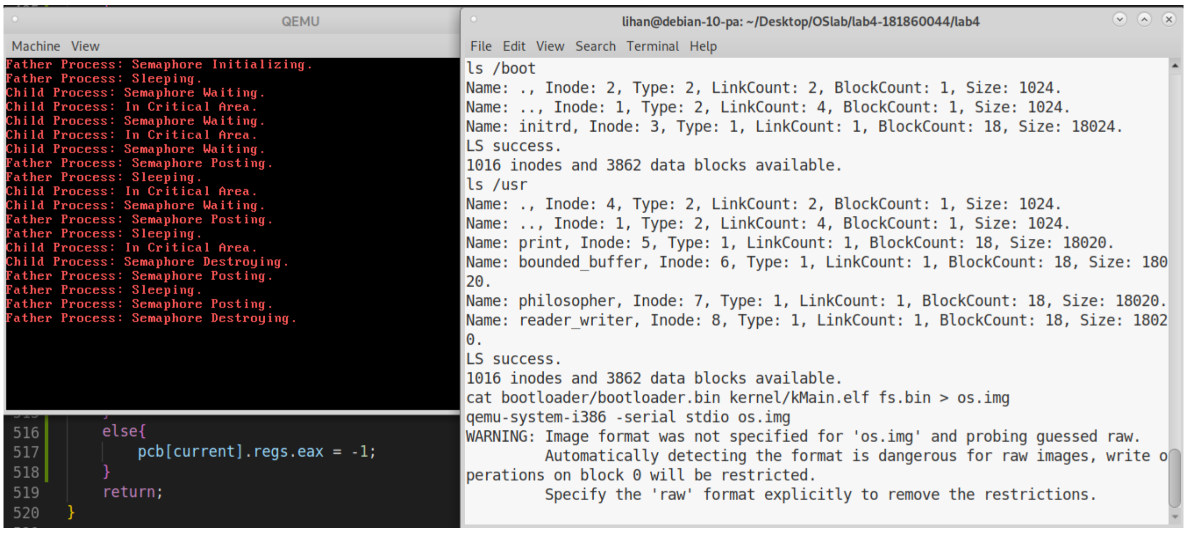
```
}

void syscallExec(struct TrapFrame *tf) {⋯
}

void syscallSleep(struct TrapFrame *tf) {⋯
```

```
ls /boot
Name: ., Inode: 2, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: initrd, Inode: 3, Type: 1, LinkCount: 1, BlockCount: 18, Size: 18024.
LS success.
1016 inodes and 3862 data blocks available.
ls /usr
Name: ., Inode: 4, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: print, Inode: 5, Type: 1, LinkCount: 1, BlockCount: 18, Size: 18020.
Name: bounded_buffer, Inode: 6, Type: 1, LinkCount: 1, BlockCount: 18, Size: 180
20.
Name: philosopher, Inode: 7, Type: 1, LinkCount: 1, BlockCount: 18, Size: 18020.
Name: reader_writer, Inode: 8, Type: 1, LinkCount: 1, BlockCount: 18, Size: 1802
0.
LS success.
1016 inodes and 3862 data blocks available.
cat bootloader/bootloader.bin kernel/kMain.elf fs.bin > os.img
qemu-system-i386 -serial stdio os.img
WARNING: Image format was not specified for 'os.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write o
perations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
```

2.3 实现信号量


QEMU
Machine View
lihan@debian-10-pa: ~/Desktop/OSlab/lab4-181860044/lab4
File Edit View Search Terminal Help

```
Father Process: Semaphore Initializing.
Father Process: Sleeping.
Child Process: Semaphore Waiting.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Child Process: In Critical Area.
Child Process: Semaphore Destroying.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Father Process: Semaphore Posting.
Father Process: Semaphore Destroying.
```

```
    else{
        pcb[current].regs.eax = -1;
    }
    return;
}
```

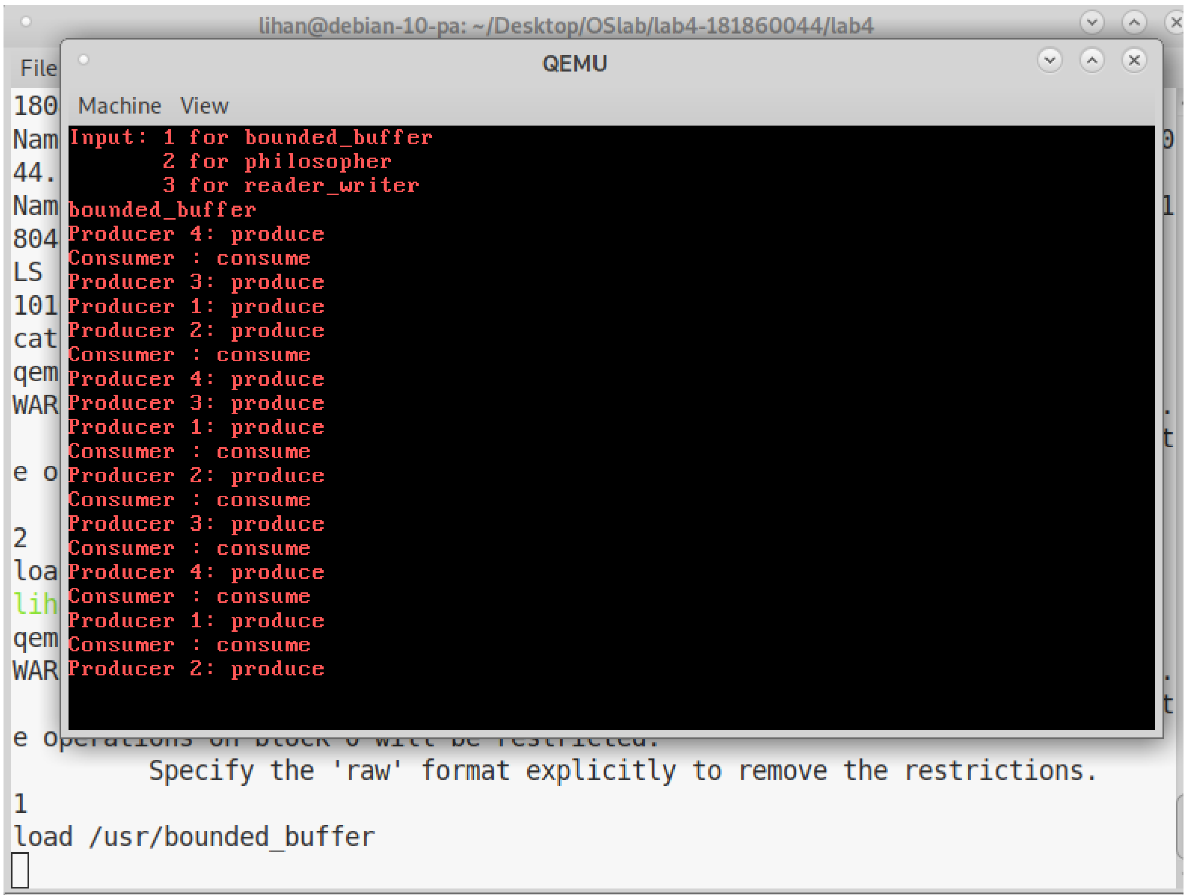
```
ls /boot
Name: ., Inode: 2, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: initrd, Inode: 3, Type: 1, LinkCount: 1, BlockCount: 18, Size: 18024.
LS success.
1016 inodes and 3862 data blocks available.
ls /usr
Name: ., Inode: 4, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: print, Inode: 5, Type: 1, LinkCount: 1, BlockCount: 18, Size: 18020.
Name: bounded_buffer, Inode: 6, Type: 1, LinkCount: 1, BlockCount: 18, Size: 180
20.
Name: philosopher, Inode: 7, Type: 1, LinkCount: 1, BlockCount: 18, Size: 18020.
Name: reader_writer, Inode: 8, Type: 1, LinkCount: 1, BlockCount: 18, Size: 1802
0.
LS success.
1016 inodes and 3862 data blocks available.
cat bootloader/bootloader.bin kernel/kMain.elf fs.bin > os.img
qemu-system-i386 -serial stdio os.img
WARNING: Image format was not specified for 'os.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write o
perations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
```

执行过程大致如下：

1. 父进程对信号量进行初始化，然后 *fork* 出一个子进程，之后进入睡眠；
2. 信号量初始值为 2，故子进程可以进入两次关键区，第三次进入时会被阻塞；
3. 父进程苏醒，将信号量释放后再次进入睡眠；
4. （被阻塞的）子进程释放，进入一次关键区，再次准备进入时被阻塞；
5. 父进程苏醒，将信号量释放后再次进入睡眠；
6. （被阻塞的）子进程释放，进入一次关键区，然后销毁信号量；
7. 父进程苏醒后执行两次 V 操作，最后销毁信号量。

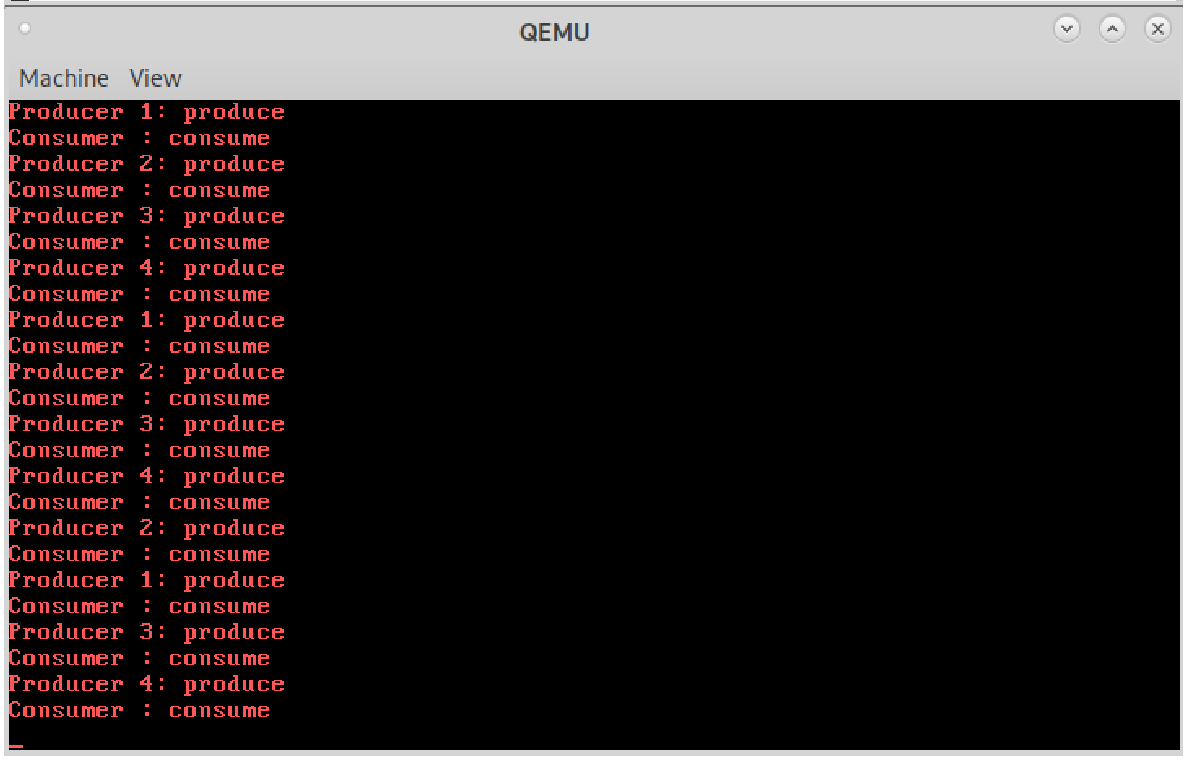
2.4 解决进程同步问题

生产者-消费者问题（*emptyBuffers* 的值初始化设置为 5）：



```
Input: 1 for bounded_buffer
       2 for philosopher
       3 for reader_writer
bounded_buffer
Producer 4: produce
Consumer : consume
Producer 3: produce
Producer 1: produce
Producer 2: produce
Consumer : consume
Producer 4: produce
Producer 3: produce
Producer 1: produce
Consumer : consume
Producer 2: produce
Consumer : consume
Producer 3: produce
Consumer : consume
Producer 4: produce
Consumer : consume
Producer 1: produce
Consumer : consume
Producer 2: produce
```

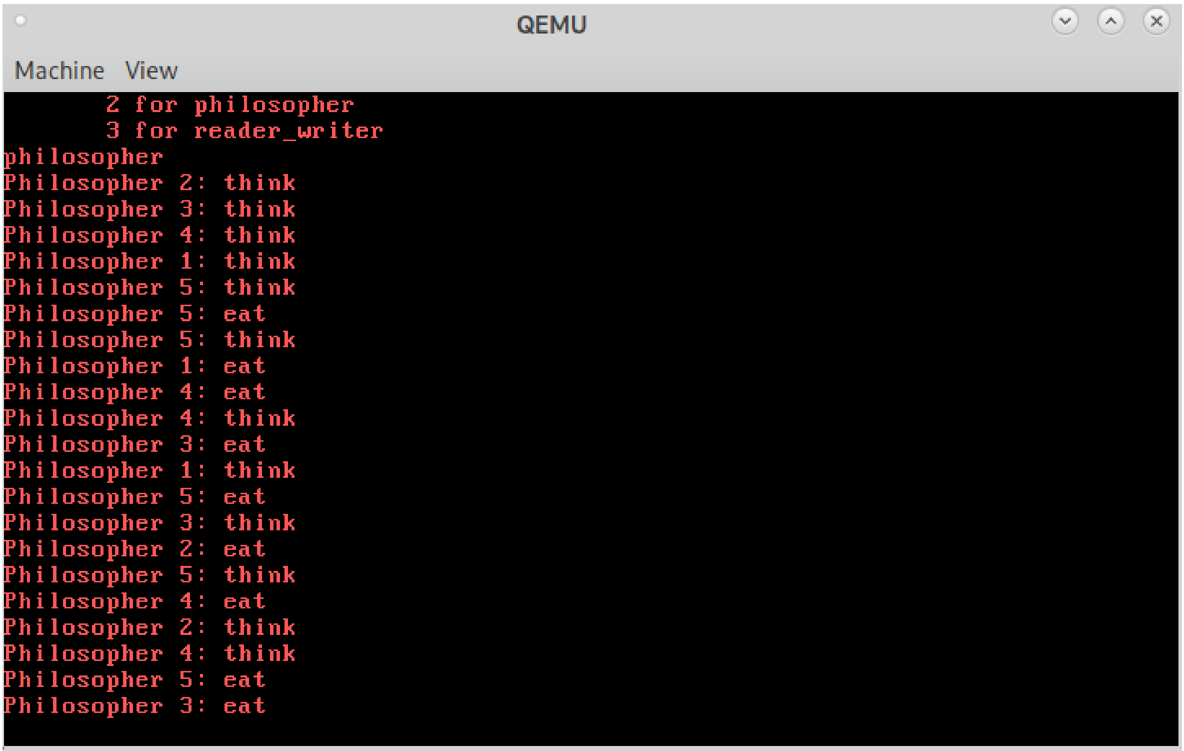
```
         Specify the 'raw' format explicitly to remove the restrictions.
1
load /usr/bounded_buffer
```



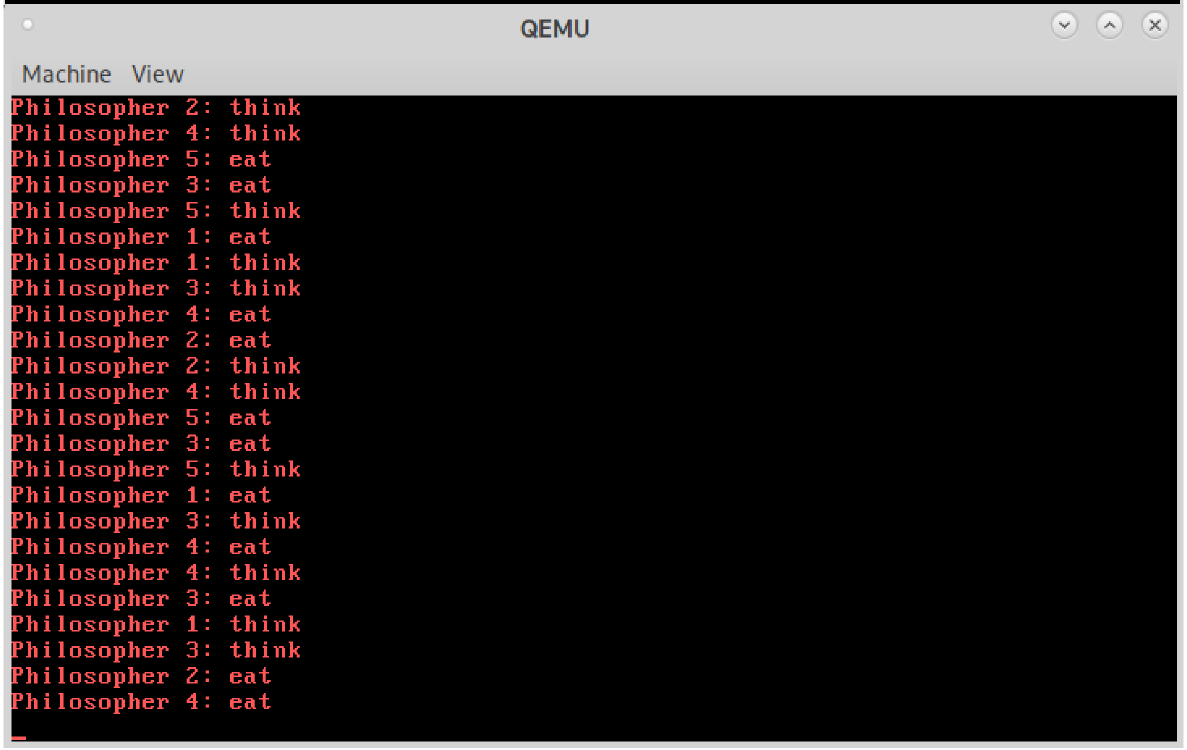
```
Producer 1: produce
Consumer : consume
Producer 2: produce
Consumer : consume
Producer 3: produce
Consumer : consume
Producer 4: produce
Consumer : consume
Producer 1: produce
Consumer : consume
Producer 2: produce
Consumer : consume
Producer 3: produce
Consumer : consume
Producer 4: produce
Consumer : consume
Producer 2: produce
Consumer : consume
Producer 1: produce
Consumer : consume
Producer 3: produce
Consumer : consume
Producer 4: produce
Consumer : consume
_
```

执行过程大致如下：4 号生产者生产一个产品 → 消费者消费一个产品 → 3、1、2号生产者一次生产一个产品 → 消费者消费一个产品 → 4、3、1号生产者依次生产一个产品（此时 *emptyBuffers* 值变为 0 ） → 消费者消费一个产品（ *emptyBuffers* 值变为 1 ） → 2 号生产者生产一个产品（ *emptyBuffers* 值变为 0 ） → ...

哲学家就餐问题：



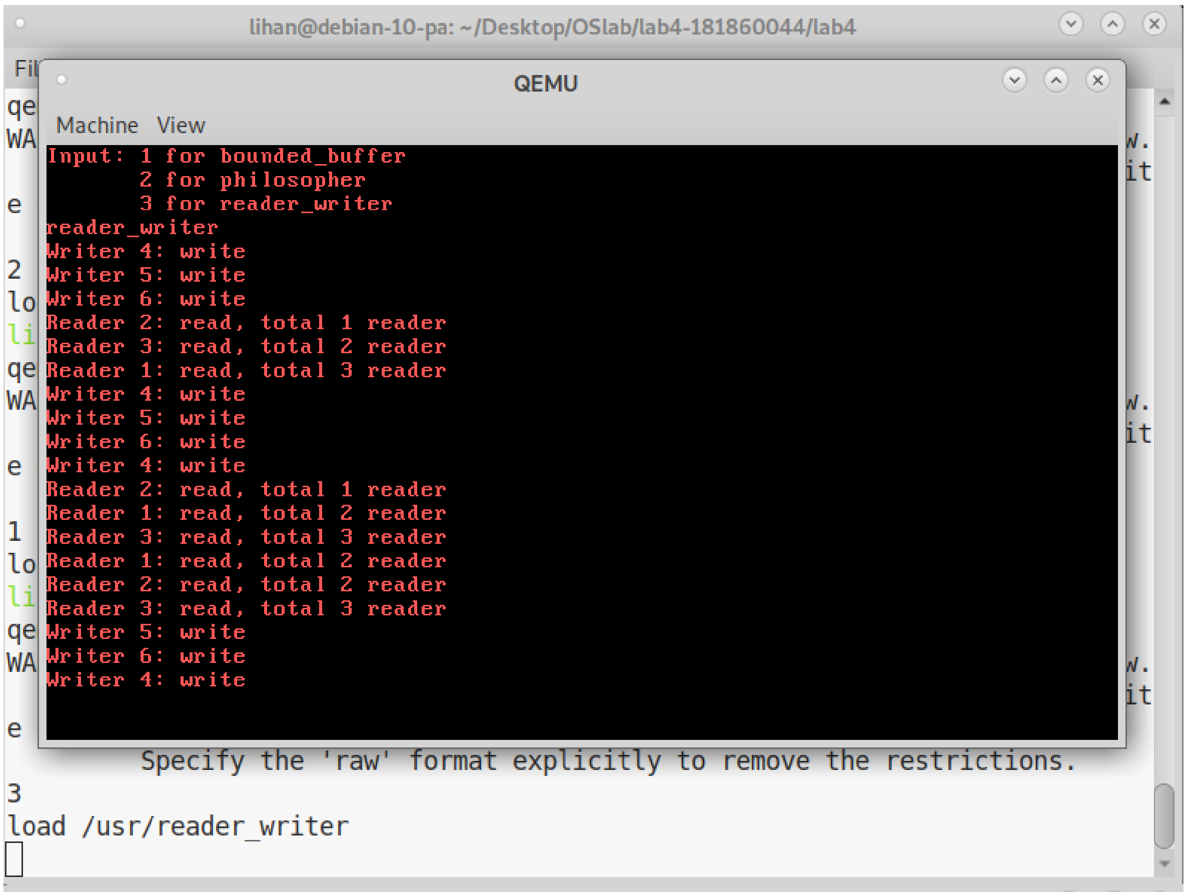
```
       2 for philosopher
       3 for reader_writer
philosopher
Philosopher 2: think
Philosopher 3: think
Philosopher 4: think
Philosopher 1: think
Philosopher 5: think
Philosopher 5: eat
Philosopher 5: think
Philosopher 1: eat
Philosopher 4: eat
Philosopher 4: think
Philosopher 3: eat
Philosopher 1: think
Philosopher 5: eat
Philosopher 3: think
Philosopher 2: eat
Philosopher 5: think
Philosopher 4: eat
Philosopher 2: think
Philosopher 4: think
Philosopher 5: eat
Philosopher 3: eat
```



```
Philosopher 2: think
Philosopher 4: think
Philosopher 5: eat
Philosopher 3: eat
Philosopher 5: think
Philosopher 1: eat
Philosopher 1: think
Philosopher 3: think
Philosopher 4: eat
Philosopher 2: eat
Philosopher 2: think
Philosopher 4: think
Philosopher 5: eat
Philosopher 3: eat
Philosopher 5: think
Philosopher 1: eat
Philosopher 3: think
Philosopher 4: eat
Philosopher 4: think
Philosopher 3: eat
Philosopher 1: think
Philosopher 3: think
Philosopher 2: eat
Philosopher 4: eat
_
```

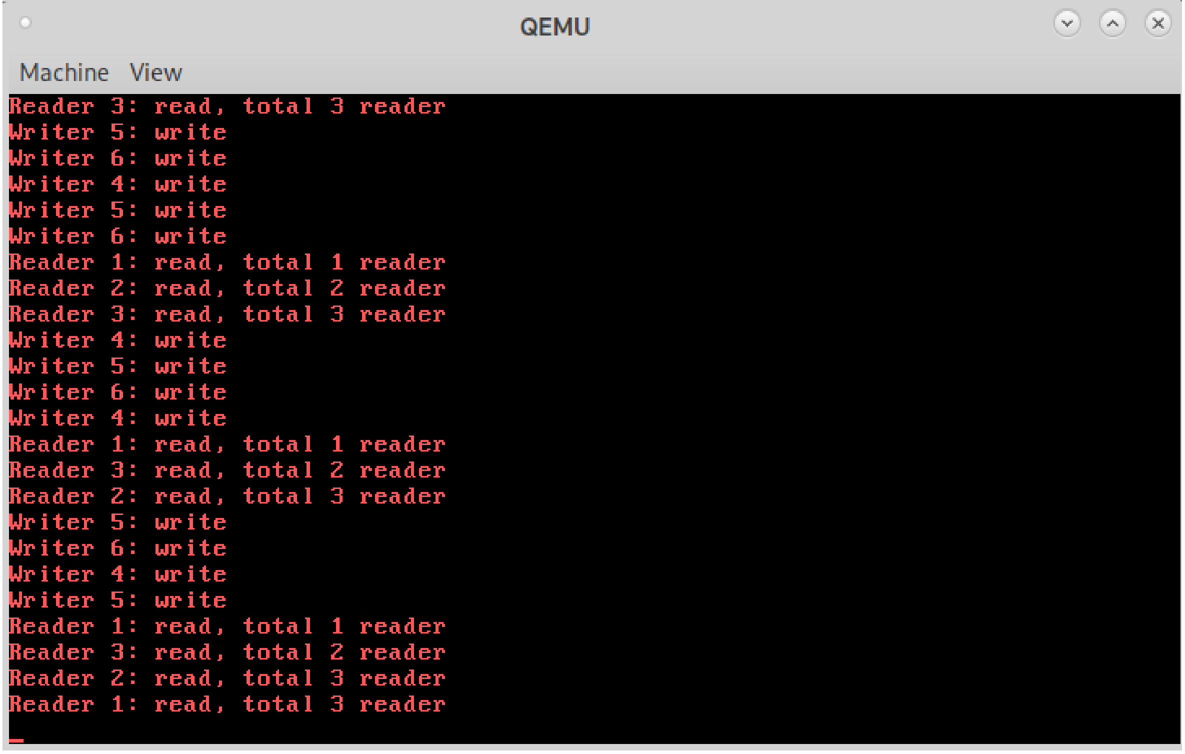
执行过程大致如下：所有哲学家都在思考 → 5 号哲学家就餐，然后进入思考 → 1、4号哲学家就餐 → 4 号哲学家进入思考(此时1号哲学家仍在就餐) → 3 号哲学家就餐 → 1 号哲学家进入思考(此时3号哲学家仍在就餐) → 5 号哲学家就餐 → 3 号哲学家进入思考(此时5号哲学家仍在就餐) → 2 号哲学家就餐 → 5 号哲学家进入思考(此时2号哲学家仍在就餐) → 4 号哲学家就餐 → 2 号哲学家进入思考(此时4号哲学家仍在就餐) → 4 号哲学家进入思考 → 5 号哲学家就餐 → 3 号哲学家就餐 → 5 号哲学家进入思考(此时3号哲学家仍在就餐) → 1 号哲学家就餐，然后进入思考 → 3 号哲学家进入思考 → ...

读者-写者问题：



```
Input: 1 for bounded_buffer
       2 for philosopher
       3 for reader_writer
reader_writer
Writer 4: write
Writer 5: write
Writer 6: write
Reader 2: read, total 1 reader
Reader 3: read, total 2 reader
Reader 1: read, total 3 reader
Writer 4: write
Writer 5: write
Writer 6: write
Writer 4: write
Reader 2: read, total 1 reader
Reader 1: read, total 2 reader
Reader 3: read, total 3 reader
Reader 1: read, total 2 reader
Reader 2: read, total 2 reader
Reader 3: read, total 3 reader
Writer 5: write
Writer 6: write
Writer 4: write
```

```
        Specify the 'raw' format explicitly to remove the restrictions.
3
load /usr/reader_writer
```



```
Reader 3: read, total 3 reader
Writer 5: write
Writer 6: write
Writer 4: write
Writer 5: write
Writer 6: write
Reader 1: read, total 1 reader
Reader 2: read, total 2 reader
Reader 3: read, total 3 reader
Writer 4: write
Writer 5: write
Writer 6: write
Writer 4: write
Reader 1: read, total 1 reader
Reader 3: read, total 2 reader
Reader 2: read, total 3 reader
Writer 5: write
Writer 6: write
Writer 4: write
Writer 5: write
Reader 1: read, total 1 reader
Reader 3: read, total 2 reader
Reader 2: read, total 3 reader
Reader 1: read, total 3 reader
_
```

执行过程大致如下：写者 4、5、6依次进行写操作 →（写完后）读者2、3、1进行读操作（3个人同时读）→（读者读完后）写者 4、5、6、4依次进行写操作 →（写完后）读者2、1、3进行读操作（3个人同时读）→ 读者 1 和 3 读完，剩下 2 在读 → 读者 1 进行读操作（2 个人同时读）→ 读者 2 读完后再次进行读操作（2 个人同事读）→ 读者 3 进行读操作（3 个人同时读）→（读者读完后）写

者 5、6、4、5、6 依次进行写操作 → （写完后）读者1、2、3进行读操作（3个人同时读）→ （读者读完后）写者 4、5、6、4 依次进行写操作 → （写完后）读者1、3、2进行读操作（3个人同时读）→ （读者读完后）写者 5、6、4、5 依次进行写操作 → ...

3. 代码修改（框架代码中TODO in lab4部分）

3.1 实现格式化输入函数

完善处理例程，在 `lab4/kernel/kernel/irqHandle.c` 中完成 `keyboardHandle` 和 `syscallReadStdIn` 。

3.2 实现进程通信

完善处理例程，相关函数为位于 `lab4/kernel/kernel/irqHandle.c` 中的 `syscallWriteShMem` 和 `syscallReadShMem` 。

3.3 实现信号量

完善处理例程，需完成 4 个子例程，即位于 `lab4/kernel/kernel/irqHandle.c` 中的 `syscallSemInit` ，`syscallSemWait` ，`syscallSemPost` 和 `syscallSemDestroy` 。

3.4 解决进程同步问题

① 在 `lab4/bounded_buffer/main.c` 中实现生产者-消费者问题
② 在 `lab4/philosopher/main.c` 中实现哲学家就餐问题
③ 在 `lab4/reader-writer/main.c` 中实现读者-写者问题
④ （选做）分别在上述文件中实现随机函数 `random`

4.思考&心得体会

1. 思考题：生产者-消费者问题中 P、V 的操作顺序有影响吗？

   有影响。对于生产者而言，如果先进行信号量 `mutex` 的 P 操作，再进行 `emptyBuffers` 的 P 操作，则当缓冲区中产品已经满时，生产者获得 `mutex` 信号量，之后将阻塞在 `emptyBuffers` 信号量上，而此时消费者尝试消费时会被阻塞在 `mutex` 信号量上，从而造成相互等待，且这种等待永远不可能结束；同样对于消费者也有这样的问题。所以 P 操作的顺序需特别注意（V 操作的次序无关紧要）。

2. 实验中在实现格式化输入函数时，测试时发现只能正确处理字母，按下数字键则会出现错误，仔细检查代码逻辑并未找到错误，最终发现是框架代码中的键盘扫描码并未对小键盘上的数字键加以处理，所以只能通过键盘上字母上方的数字键进行输入。

3. 在实现生产者-消费者问题之前，先对是否成功 *fork* 出 5 个进程进行测试，然后发现确是五个进程，但是其中有两个进程的进程号相同，最后发现是在进行 *fork* 之后执行一次 *printf* 后就将进程销毁，导致还没生成五个进程之前就有某个进程已经被销毁了，从而出现了相同的进程号。

4. 在读者-写者问题中，意识到若将 *Rcount* 定义成信号量则不便于获取其值，而仅仅将其定义成一个计数变量则会由于不同进程之间并不共享其值而导致程序运行出错，于是将其放在共享内存中，每次使用到 *Rcount* 时先从共享内存中读取，若发生修改则将其写回共享内存。

5. 此次实验使本人对基于共享内存实现进程通信的机制以及基于信号量的进程同步机制有了进一步的了解，同时也在一定程度上提升了编写进程同步代码以及调试多进程代码的能力。