

# Lab4 实验报告

221220113

## 4.1 完善 scanf

在 lib/syscall.c 中找到 scanf 的定义，其间接调用了需要实现的 syscallReadStdIn。

根据 scanf 中调用 syscall 的方法，对对应的参数做操作即可。

需要注意的是，如果 Device[STD\_IN].value == 0，说明没有输入需要阻塞，相应的需要

在 keyboardHandle 中增加一个唤醒操作。

最终 4.1 测试如下：



```
QEMU - Press Ctrl-Alt to exit mouse grab
input: "Test %c Test %6s %d %x"
Ret: 4; a, oslab, 0, adc.
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.
```

## 4.2 实现信号量

完善 irqHandle 中的相关函数即可，测试结果如上，分析：

Sem 初始化 value=2，子进程首先进行 wait，子进程 wait 第三次时 value<0，阻塞等待

parent post。接下来 parent post，child wait 依次进行两次，然后 child 进程结束，

parent 独占时间片进行两次 post，结束。

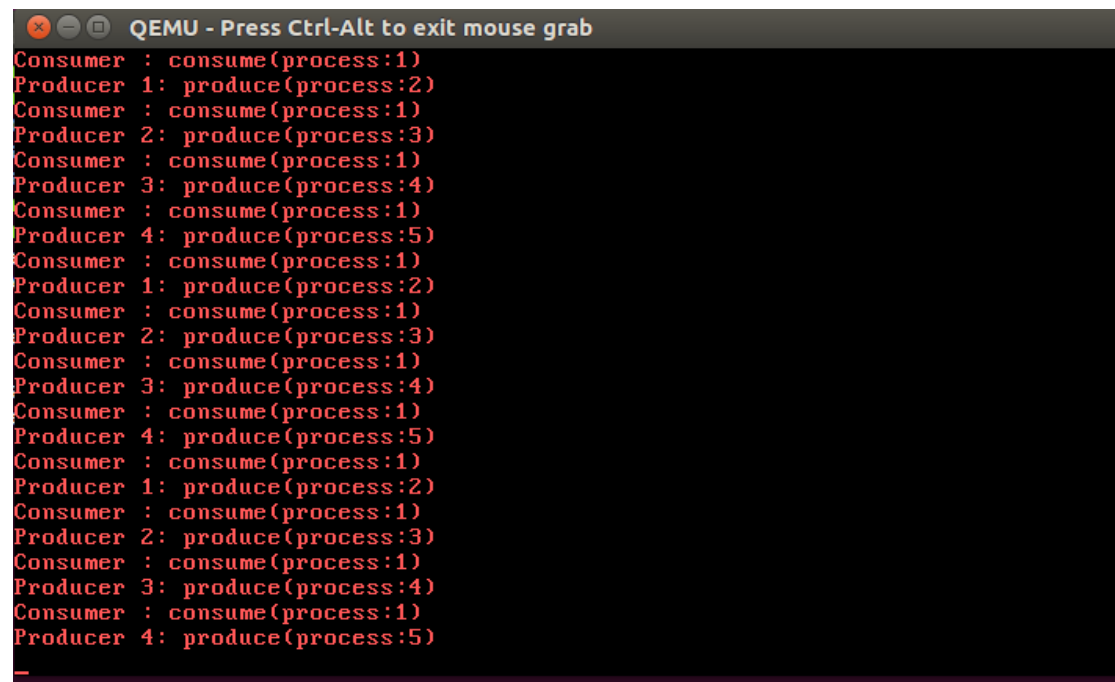
## 4.3 完成生产者消费者问题

本次实验不要求实际操作缓冲区，只需要输出一段文字即可，所以并不实际需要 mutex，

但是还是保留下来 mutex，以表示实际操作缓冲区时的临界区。

将缓冲区大小设为 8，即 full\_buffers 初始化 0，empty\_buffers 初始化 8。

测试如下：

A screenshot of a QEMU terminal window. The title bar reads "QEMU - Press Ctrl-Alt to exit mouse grab". The terminal displays a sequence of log messages in red text on a black background. The messages are: "Consumer : consume(process:1)", "Producer 1: produce(process:2)", "Consumer : consume(process:1)", "Producer 2: produce(process:3)", "Consumer : consume(process:1)", "Producer 3: produce(process:4)", "Consumer : consume(process:1)", "Producer 4: produce(process:5)", "Consumer : consume(process:1)", "Producer 1: produce(process:2)", "Consumer : consume(process:1)", "Producer 2: produce(process:3)", "Consumer : consume(process:1)", "Producer 3: produce(process:4)", "Consumer : consume(process:1)", "Producer 4: produce(process:5)", "Consumer : consume(process:1)", "Producer 1: produce(process:2)", "Consumer : consume(process:1)", "Producer 2: produce(process:3)", "Consumer : consume(process:1)", "Producer 3: produce(process:4)", "Consumer : consume(process:1)", and "Producer 4: produce(process:5)". The output shows a strict alternation between consumer and producer actions.

```
Consumer : consume(process:1)
Producer 1: produce(process:2)
Consumer : consume(process:1)
Producer 2: produce(process:3)
Consumer : consume(process:1)
Producer 3: produce(process:4)
Consumer : consume(process:1)
Producer 4: produce(process:5)
Consumer : consume(process:1)
Producer 1: produce(process:2)
Consumer : consume(process:1)
Producer 2: produce(process:3)
Consumer : consume(process:1)
Producer 3: produce(process:4)
Consumer : consume(process:1)
Producer 4: produce(process:5)
Consumer : consume(process:1)
Producer 1: produce(process:2)
Consumer : consume(process:1)
Producer 2: produce(process:3)
Consumer : consume(process:1)
Producer 3: produce(process:4)
Consumer : consume(process:1)
Producer 4: produce(process:5)
```

截图了比较靠后的输出。

实际上，在开头处还有连续 8 个 produce。