

# OS Lab4 实验报告

---

## 实验信息

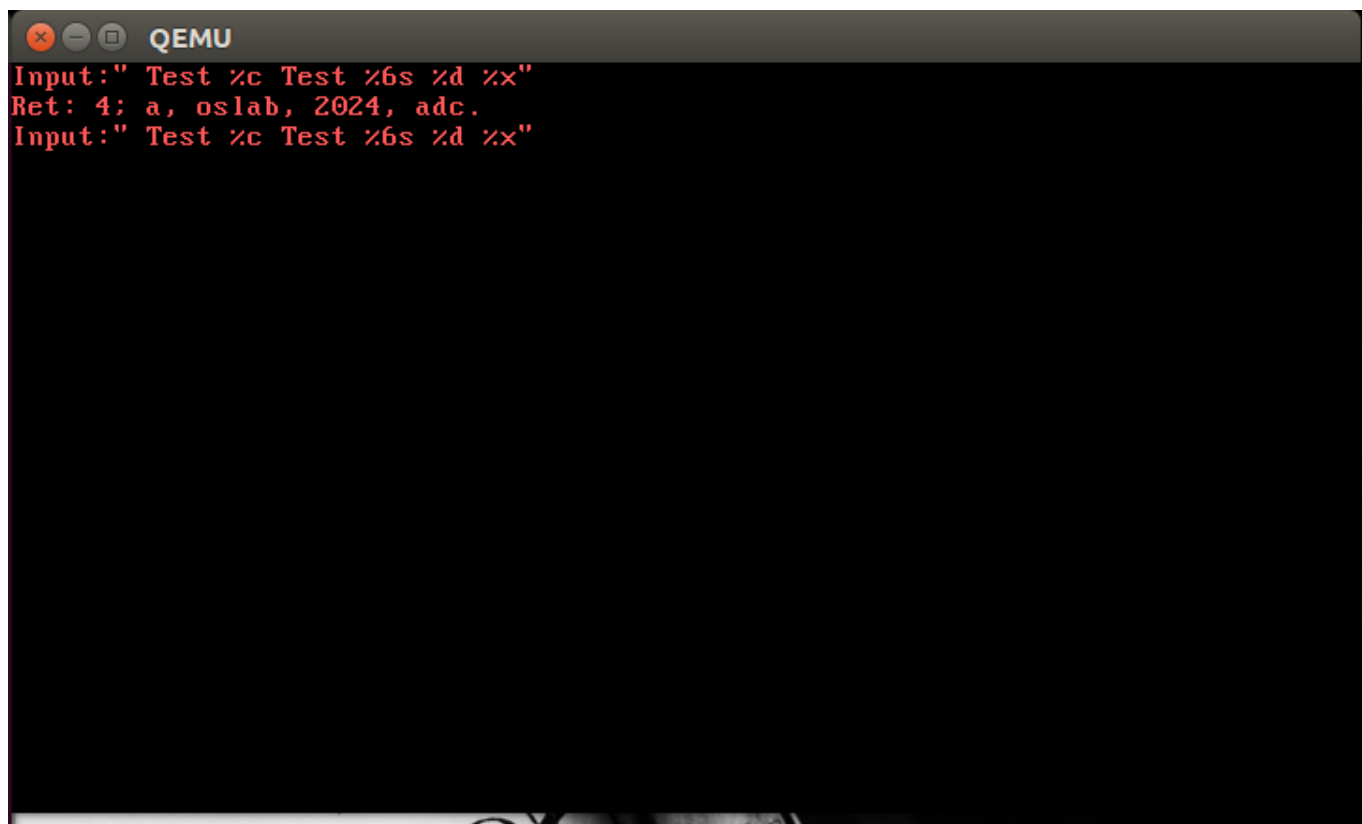
---

- 姓名：王旭
- 学号：221220034
- 邮箱：[2069625874@qq.com](mailto:2069625874@qq.com)
- 实验进度：已全部完成，并且完成哲学家、读写问题的测试。

## 实验结果

---

### 4.1



```
QEMU
Input:" Test %c Test %6s %d %x"
Ret: 4; a, oslab, 2024, adc.
Input:" Test %c Test %6s %d %x"
```

### 4.2

```
QEMU
Input: " Test %c Test %6s %d %x"
Ret: 4; a, oslab, 2024, 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.
_
```

## 结果分析

先是父进程运行，打印前两行，然后子进程进入临界区两次后阻塞，然后依次运行父进程、子进程，因为这时sem的值在-1和0变化，父进程sleep4次后，i减为0，父进程exit，子进程也随之销毁，程序结束。

## 4.3

### 生产者-消费者问题

- 背景：4个生产者、1个消费者，我选择缓冲区有6个空闲位置，消费者消费一个后sleep。

```
QEMU
Producer 2: Producing.
Producer 3: Producing.
Producer 4: Producing.
Producer 5: Producing.
Consumer : Consuming.
Producer 2: Producing.
Producer 3: Producing.
Producer 4: Producing.
Consumer : Consuming.
Producer 5: Producing.
Consumer : Consuming.
Producer 2: Producing.
Consumer : Consuming.
Producer 3: Producing.
Consumer : Consuming.
Producer 4: Producing.
Consumer : Consuming.
Producer 5: Producing.
Consumer : Consuming.
```

结果与预想一致，先是2、3、4、5生产者进程运行，然后消费者，buffer中此时有三个空闲位置，2、3、4依次生产，然后5阻塞，消费者消费，下面开始轮流。

## 哲学家问题

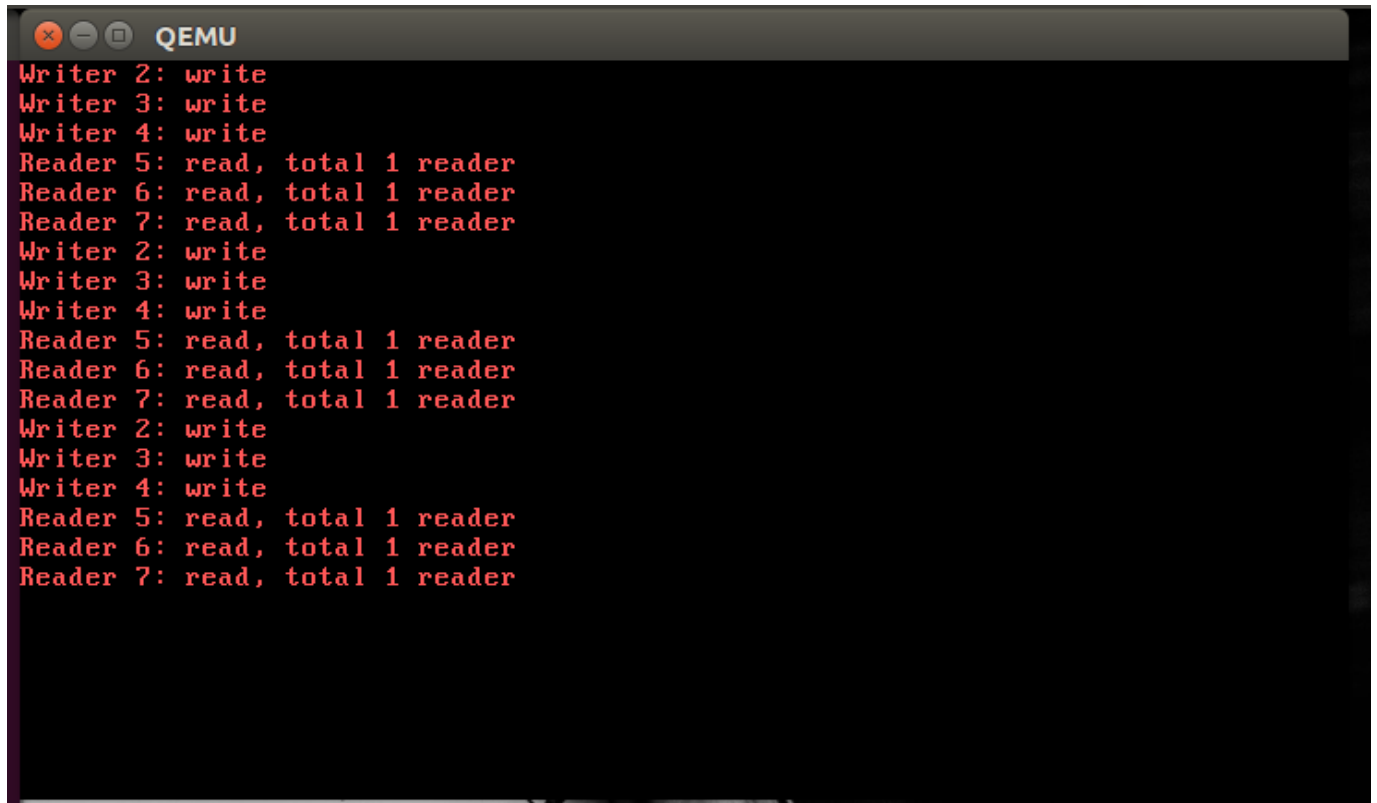
- 背景：5为哲学家，5把叉子，按照手册上第三个实现方式实现，没有死锁风险。

```
QEMU
Philosopher 2: think
Philosopher 2: eat
Philosopher 3: think
Philosopher 3: eat
Philosopher 4: think
Philosopher 4: eat
Philosopher 5: think
Philosopher 5: eat
Philosopher 6: think
Philosopher 6: eat
Philosopher 2: think
Philosopher 2: eat
Philosopher 3: think
Philosopher 3: eat
Philosopher 4: think
Philosopher 4: eat
Philosopher 5: think
Philosopher 5: eat
Philosopher 6: think
Philosopher 6: eat
```

基本按照顺序依次运行进程。

## 读写问题

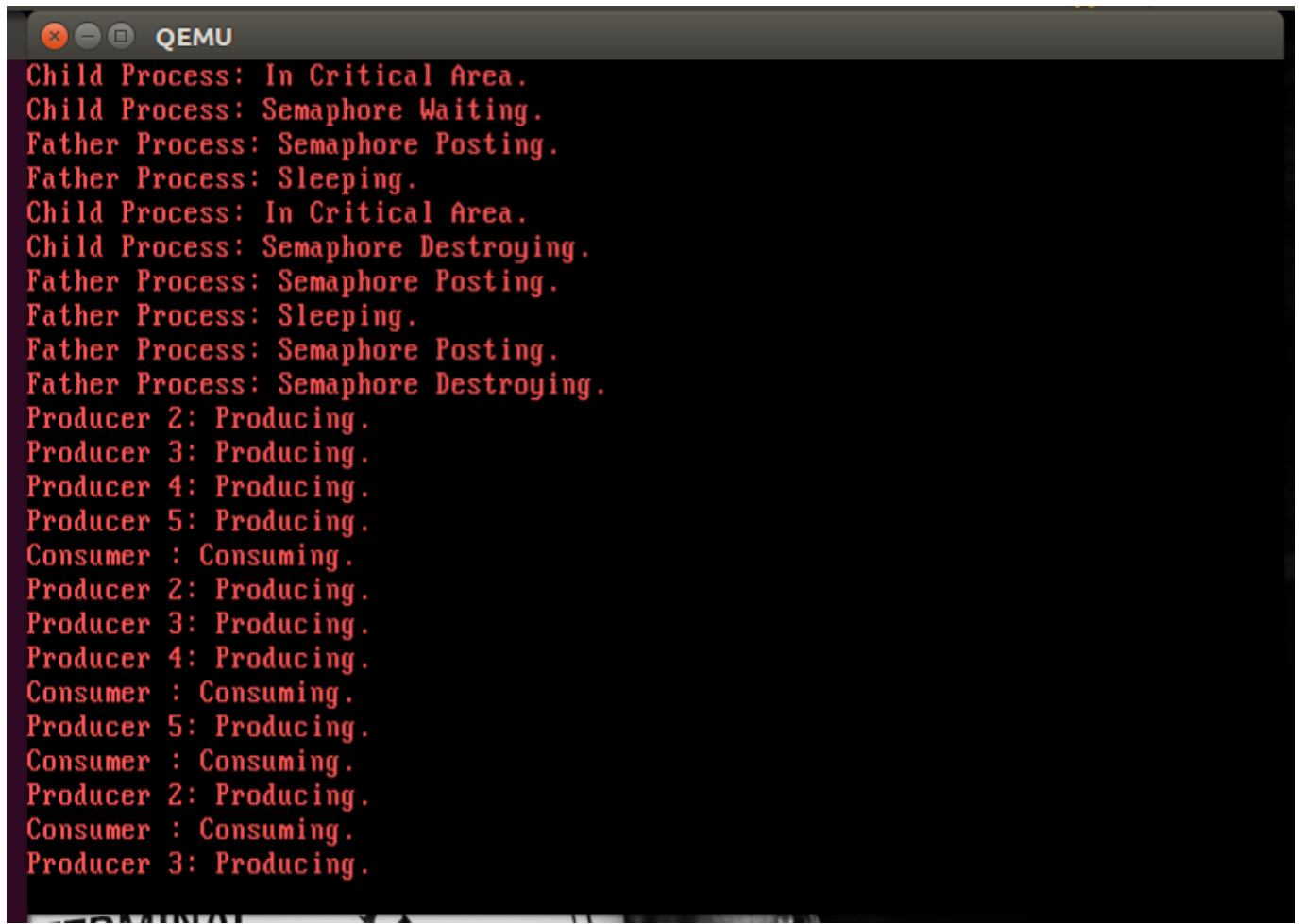
- 背景：3个读进程、3个写进程，采取读者优先。



```
QEMU
Writer 2: write
Writer 3: write
Writer 4: write
Reader 5: read, total 1 reader
Reader 6: read, total 1 reader
Reader 7: read, total 1 reader
Writer 2: write
Writer 3: write
Writer 4: write
Reader 5: read, total 1 reader
Reader 6: read, total 1 reader
Reader 7: read, total 1 reader
Writer 2: write
Writer 3: write
Writer 4: write
Reader 5: read, total 1 reader
Reader 6: read, total 1 reader
Reader 7: read, total 1 reader
```

由于顺序fork写进程、读进程，没有抢夺策略，所以写进程会依次执行完，才轮到pcb在后面的读进程。

## 总的实验结果

A screenshot of a QEMU terminal window with a black background and red text. The window title bar shows standard Linux window controls and the text 'QEMU'. The output shows a sequence of process actions: '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.', 'Producer 2: Producing.', 'Producer 3: Producing.', 'Producer 4: Producing.', 'Producer 5: Producing.', 'Consumer : Consuming.', 'Producer 2: Producing.', 'Producer 3: Producing.', 'Producer 4: Producing.', 'Consumer : Consuming.', 'Producer 5: Producing.', 'Consumer : Consuming.', 'Producer 2: Producing.', 'Consumer : Consuming.', and 'Producer 3: Producing.'.

```
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.
Producer 2: Producing.
Producer 3: Producing.
Producer 4: Producing.
Producer 5: Producing.
Consumer : Consuming.
Producer 2: Producing.
Producer 3: Producing.
Producer 4: Producing.
Consumer : Consuming.
Producer 5: Producing.
Consumer : Consuming.
Producer 2: Producing.
Consumer : Consuming.
Producer 3: Producing.
```

将4.2中父进程的`exit ()`删除后，一切正常。

## 实验主要代码

### 4.1

- `void keyboardHandle(struct StackFrame *sf)`

此处添加了处理`dev[STD_IN].value<0`的处理，就是释放一个该设备上阻塞的进程。

- `void syscallReadStdIn(struct StackFrame *sf)`

此处分为`dev[STD_IN].value`是否大于0的两种情况，若小于0，则将-1复制给当下进程的`eax`，如果不是，则阻塞当下进程，并且调用时钟中断，等到恢复运行后，开始读缓冲区，并记录读的个数。，赋值给`eax`。

### 4.2

- `syscallSemInit()`

初始化一个信号量，在se[MAX\_SEM\_NUM]中顺序查找一个空闲的信号量，未找到的话，eax返回-1，找到的话，赋值values、state变为1、初始化其pcb链表、返回sem的序号。

- syscallSemWait()

先判断是否是非法调用，如sem序号无效、该信号量是空闲的，若是，返回-1。

若不是，则value-1、返回0；若values<0,当下进程阻塞，并且放在该信号量的等待进程链表中、调用时钟中断。

- syscallSemPost()

先判断是否是非法调用，如sem序号无效、该信号量是空闲的，若是，返回-1。

将sem信号量的value+1、如果value<=0,释放一个进程、返回0。

- syscallSemDestroy()

先判断是否非法；

不非法的话，信号量的state赋值为0，eax赋值为0，调用时钟中断。

## 4.3

app/main.c文件：

- 生产者-消费者问题：

```
void producer_consume()
```

消费者：

```
while(1){  
    int id = getpid();  
    sem_wait(&empty); //是否有产品  
    // sleep(128);  
    sem_wait(&mutex); //互斥锁  
    printf("Producer %d: Producing.\n", id);  
    sleep(128);  
    sem_post(&mutex);  
    // sleep(128);  
    sem_post(&full);  
    sleep(128);  
}
```

生产者：

```

while(1) //消费者
{
    sem_wait(&full); //是否有空闲位置
    sem_wait(&mutex); //互斥锁
    printf("Consumer : Consuming.\n");
    sleep(128);
    sem_post(&mutex);
    sem_post(&empty);
    sleep(128);
}

```

- 哲学家问题：

```
void Philosophers()
```

偶数序号的哲学家先拿右手边的叉子，奇数序号的哲学家先拿左手边的叉子，可避免死锁。

- 读写问题：

```
void Read_Write()
```

先创建3个写者进程，再创建3个读者进程，选择在第一个读进程开始后，写进程就不能访问临界区，直至最后一个读进程完成，方可进行写进程。