Lab4 实验报告

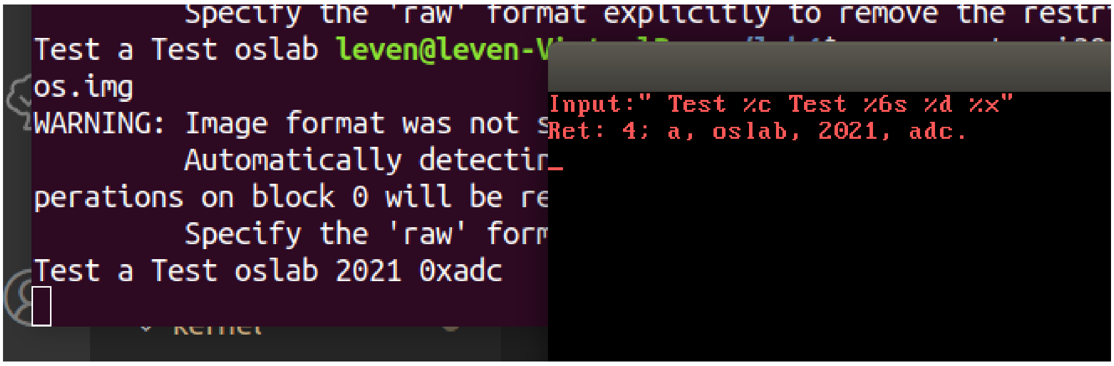
南京大学 人工智能学院

191300036 刘晔闻 450008668@qq.com

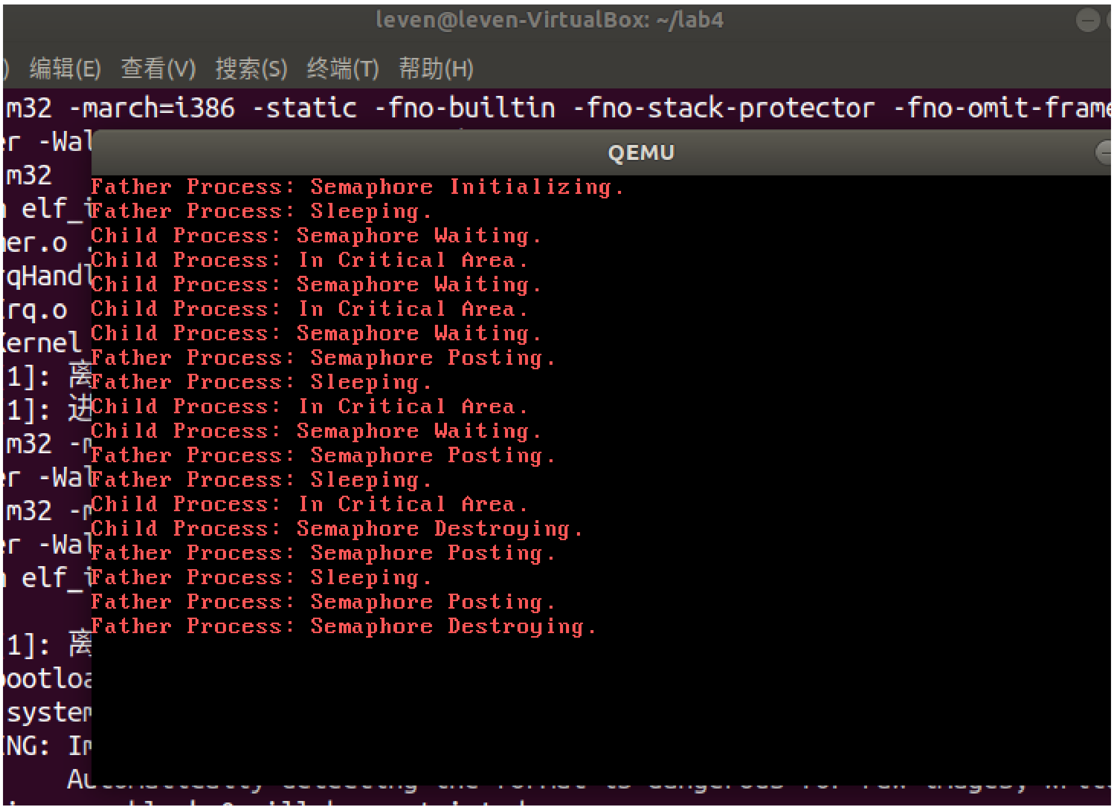
实验进度：我完成了所有实验内容，包括必做和选做。

实验结果：

1. scanf：



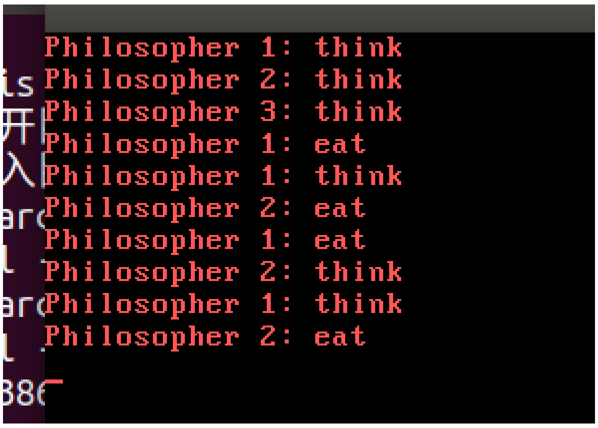
2. 信号量：



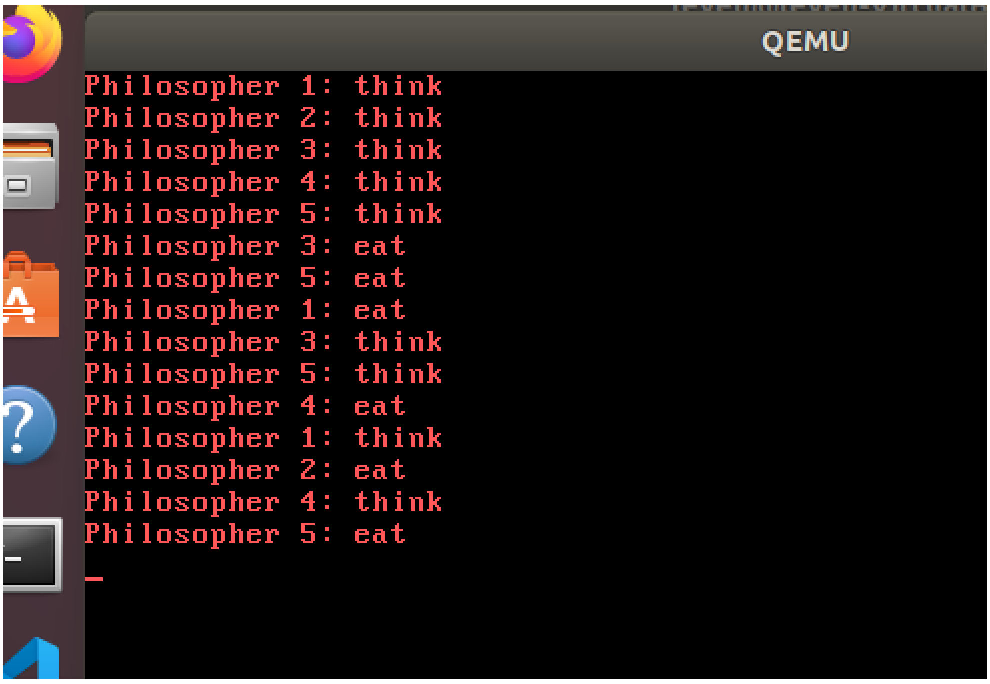
3. 进程同步：

（1） 哲学家就餐：

一开始：

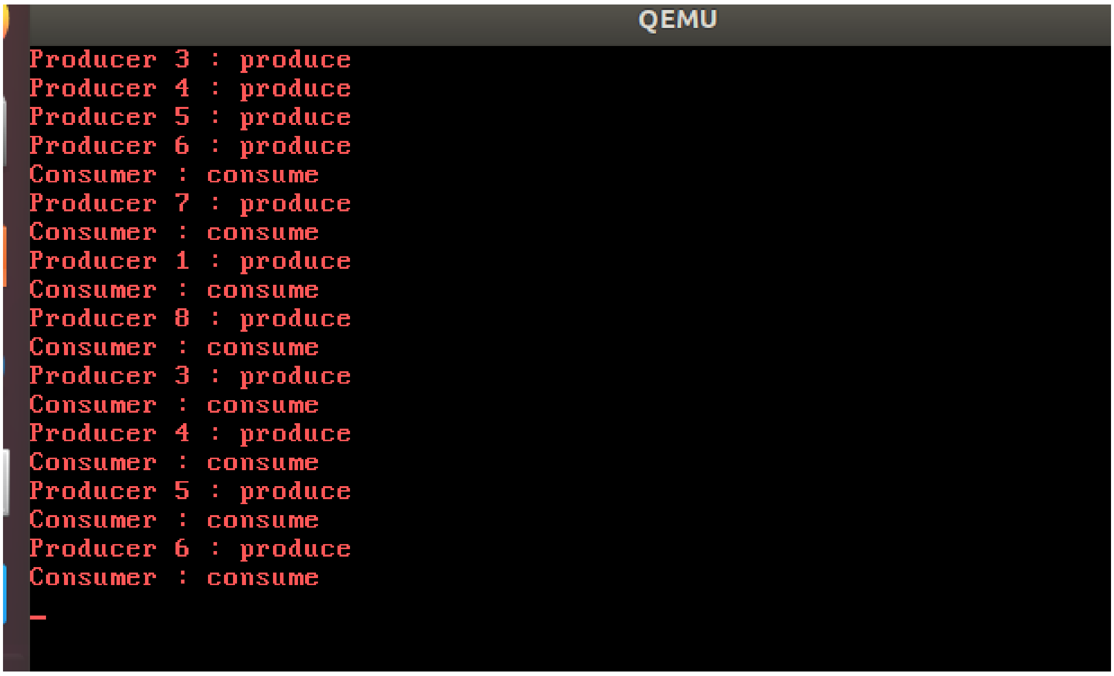


后来调整了框架代码的线程数：

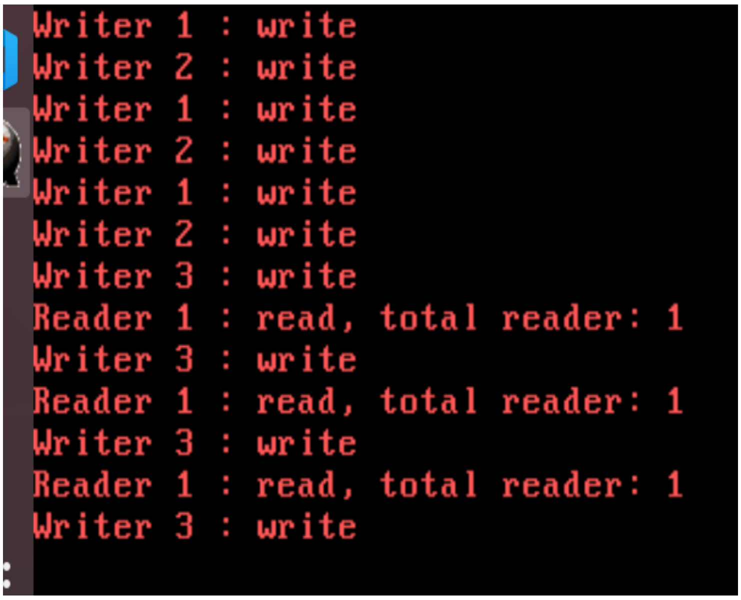


（2）　生产者消费者：

getpid（）得到的线程 id 可能还有一些问题；



（3）读者写者：



实验过程：

1. **keyboardHandle：** 首先是 keyboardHandle 的作用，

    1. 将读取到的keyCode放入到keyBuffer中
    2. 唤醒阻塞在dev[STD_IN]上的一个进程

    其次，根据实验指南的提示

    以下代码可以从信号量i上阻塞的进程列表取出一个进程：

```
pt = (ProcessTable*)((uint32_t)(sem[i].pcb.prev) -
             (uint32_t)&(((ProcessTable*)0)->blocked));
sem[i].pcb.prev = (sem[i].pcb.prev)->prev;
(sem[i].pcb.prev)->next = &(sem[i].pcb);
```

    可以简单地实现 keyboardHandle 中的 TODO 内容。

2. syscallReadStdIn：它的作用是

    1. 如果dev[STD_IN].value == 0，将当前进程阻塞在dev[STD_IN]上
    2. 进程被唤醒，读keyBuffer中的所有数据

    Step 1：使用实验指南的提示阻塞 value=1 时的进程；

    这样将current线程加到信号量i的阻塞列表可以通过以下代码实现

```
pcb[current].blocked.next = sem[i].pcb.next;
pcb[current].blocked.prev = &(sem[i].pcb);
sem[i].pcb.next = &(pcb[current].blocked);
(pcb[current].blocked.next)->prev = &(pcb[current].blocked);
```

Step 2：读取 keybuffer 中的数据；

和实验2中printf的处理例程类似，以下代码可以将读取的字符character传到用户进程

```
int sel = sf->ds;
char *str = (char *)sf->edx;
int i = 0;
asm volatile("movw %0, %%es"::"m"(sel));
asm volatile("movb %0, %%es:(%1)"::"r"(character),"r"(str + i));
```

注意，只有在 character 不为 0 且 buffer 不为空时（tail ＞ head）才能读取。

进行取余操作，防止 tail ＜ head，出现溢出问题。

```
int size = (bufferTail + MAX_KEYBUFFER_SIZE - bufferHead) % MAX_KEYBUFFER_SIZE;
```

```
if(character != 0){
    asm volatile("movb %0, %%es:(%1)"::"r"(character),"r"(str+i));
```

并且，测试 scanf 时输入要慢一点，因为 IO 设备进入缓冲区的速度不同，输入过快可能导致冲突，提前结束输入。

3. syscallSemInit：

sem_init系统调用用于初始化信号量，其中参数value用于指定信号量的初始值，初始化成功则返回0，指针sem指向初始化成功的信号量，否则返回-1

State 值 0 表示未使用，1 表示正在使用；

并且 kvm.c 中有 initsem 函数，初始化 sem；类似可以实现 syscallseminit 函数；

```
void initSem() {
    int i;
    for (i = 0; i < MAX_SEM_NUM; i++) {
        sem[i].state = 0; // 0: not in use; 1: in use;
        sem[i].value = 0; // >=0: no process blocked; -1: 1 process blocked; -2: 2
        sem[i].pcb.next = &(sem[i].pcb);
        sem[i].pcb.prev = &(sem[i].pcb);
    }
}
```

4. syscallSemWait：

sem_wait系统调用对应信号量的P操作，其使得sem指向的信号量的value减一，若value取值小于0，则阻塞自身，否则进程继续执行，若操作成功则返回0，否则返回-1

阻塞进程的代码类似 syscallReadStdIn 中 step1 的代码；

Sleeptime = -1 表示一个进程被阻塞；

5. syscallSemPost：

sem_post系统调用对应信号量的V操作，其使得sem指向的信号量的value增一，若value取值不大于0，则释放一个阻塞在该信号量上进程（即将该进程设置为就绪态），若操作成功则返回0，否则返回-1

唤醒阻塞进程代码类似 keyboardHandle；

6. syscallSemDestroy：

sem_destroy系统调用用于销毁sem指向的信号量，销毁成功则返回0，否则返回-1，若尚有进程阻塞在该信号量上，可带来未知错误

7. syscallGetPid：

返回当前进程的 pid；

```
void syscallGetPid(struct StackFrame *sf) {
    sf -> eax = current;
    return;
```

8. 哲学家就餐：

- 5个哲学家同时运行
- 哲学家思考，`printf("Philosopher %d: think\n", id);`
- 哲学家就餐，`printf("Philosopher %d: eat\n", id);`
- 任意P、V及思考、就餐动作之间添加`sleep(128);`

9. 生产在消费者问题：

生产者-消费者问题：

- 4个生产者，1个消费者同时运行
- 生产者生产，`printf("Producer %d: produce\n", id);`
- 消费者消费，`printf("Consumer : consume\n");`
- 任意P、V及生产、消费动作之间添加`sleep(128);`

10. 读者写者问题：

读者-写者问题：

- 3个读者，3个写者同时运行
- 读者读数据，`printf("Reader %d: read, total %d reader\n", id, Rcount);`
- 写者写数据，`printf("Writer %d: write\n", id);`
- 任意P、V及读、写动作之间添加`sleep(128);`

11. 调整并行线程数：

完成哲学家用餐问题是发现一开始只有三个哲学家线程同时运行，之后发现是框架代码线程数的限制；

```
#define NR_SEGMENTS     20          // GDT size   // XXXlimit 10
```

```
#define MAX_PCB_NUM ((NR_SEGMENTS-2)/2)   //XXX limit 4
```

思考题：

1. 有没有更好的方式处理这个就餐问题？

可以判断手中有筷子的哲学家人数 n，当 n=4 时，第五位哲学家将不能拿筷子，这样也可以避免死锁。

2. P、V 的操作顺序有影响吗？

P、V 操作的顺序有影响，因为 empty 和 full 的初始值不同。