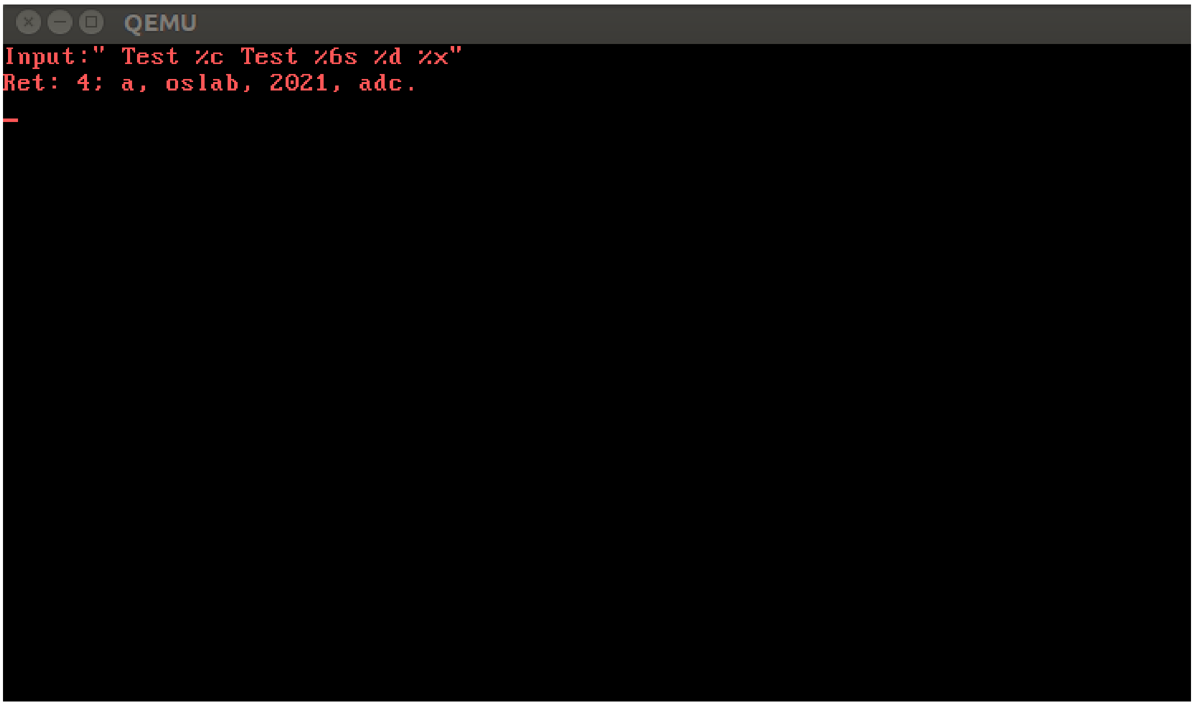
lab4 实验报告

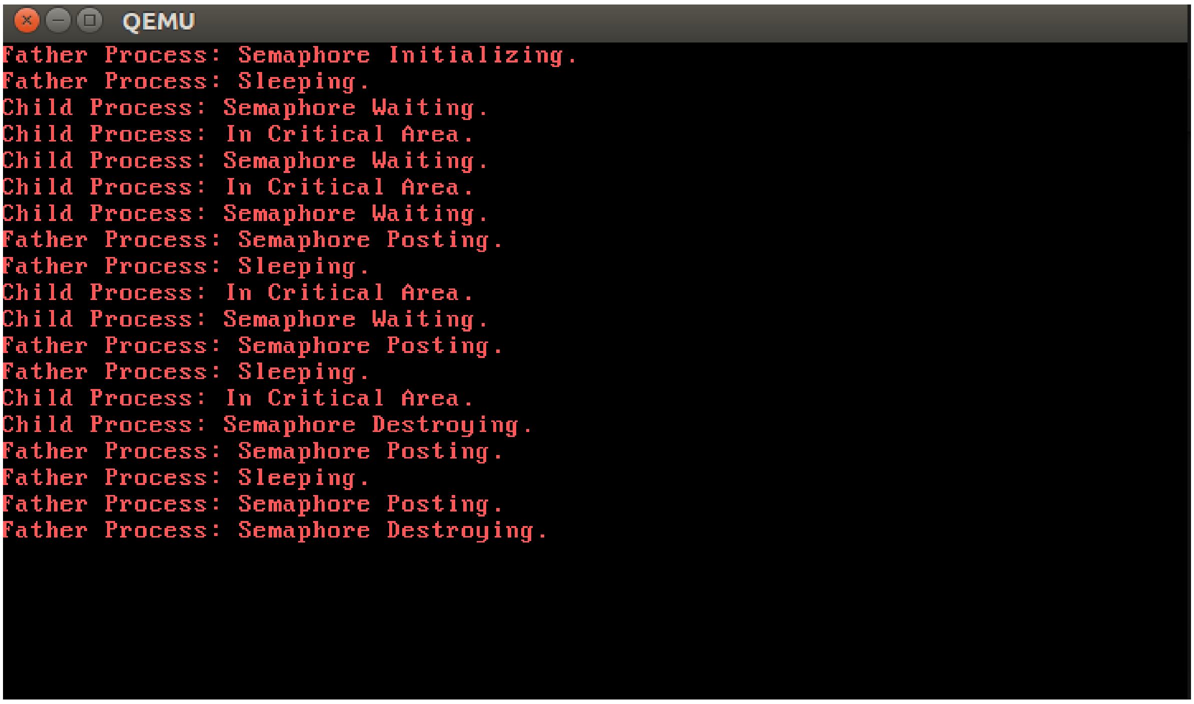
李培佳 191300029 191300029@smail.nju.edu.cn

1.实验进度

完成了所有任务。

2.实验结果





```
output
qemu-system-i386 -serial stdio os.in
Philosopher 1: think
Philosopher 2: think
Philosopher 3: think
Philosopher 4: think
Philosopher 5: think
Philosopher 1: eat
Philosopher 4: eat
Philosopher 1: think
Philosopher 2: eat
Philosopher 4: think
Philosopher 5: eat
Philosopher 2: think
Philosopher 3: eat
Philosopher 5: think
Philosopher 1: eat
Philosopher 3: think
Philosopher 4: eat
Philosopher 1: think
Philosopher 2: eat
Philosopher 4: think
Philosopher 5: eat
Philosopher 2: think
Philosopher 3: eat
Philosopher 5: think
Philosopher 1: eat
Philosopher 3: think
Philosopher 4: eat
Philosopher 1: think
Philosopher 2: eat
```

lab 4.2 结果描述

从`app/mian.c`的源码来看，首先为父进程初始化信号量，`value`设置为2，随后创建子进程。

此时继续执行父进程

```
while( i != 0) {
    i --;
    printf("Father Process: Sleeping.\n");
    sleep(128);
    printf("Father Process: Semaphore Posting.\n");
    sem_post(&sem);
}
```

打印`Father Process: Sleeping.`，父进程休眠，进入子进程

```
while( i != 0) {
    i --;
    printf("Child Process: Semaphore Waiting.\n");
    sem_wait(&sem);
    printf("Child Process: In Critical Area.\n");
}
printf("Child Process: Semaphore Destroying.\n");
sem_destroy(&sem);
exit();
```

由于信号量的值为 2,子进程可以进入两次关键区，即`Critical Area`，在第三次时被阻塞，父进程苏醒，释放信号量，又去睡眠，接着执行子进程，进入一次`Critical Area`，然后被阻塞，等待父进程苏醒，释放信号量，再进入`Critical Area`，此时`i`被减为0，故子进程的信号量被销毁，等待父进程的`i`也减为0，父进程销毁信号量。

3.修改的代码位置

3.1 格式化输入函数

`scanf`已经给出，需要实现的是`syscallReadStdIn`和`keyboardHandle`的同步

需要考虑进程同步的地方在于`dev[STD_IN]`，框架代码中将`STD_IN`抽象成了`Device`，也是一个信号量，在执行到`syscallReadStdIn`后，我们要对`dev[STD_IN]`进行一个`P`操作，即为了确保键盘不会被别的进程使用，如果`dev[STD_IN]`已经被占用，则阻塞当前进程，即将当前进程放在`dev[STD_IN]`的阻塞队列上。具体代码如下：

```
dev[STD_IN].value--;

pcb[current].blocked.next = dev[STD_IN].pcb.next;
pcb[current].blocked.prev = &(dev[STD_IN].pcb);
dev[STD_IN].pcb.next = &(pcb[current].blocked);
(pcb[current].blocked.next)->prev = &(pcb[current].blocked);

pcb[current].state = STATE_BLOCKED;
asm volatile("int $0x20");
```

进入`keyboardHandle`后，执行一个`V`操作，释放`dev[STD_IN]`上被阻塞的进程，即`syscallReadStdIn`中被阻塞的进程

```
dev[STD_IN].value++;

ProcessTable *pt = (ProcessTable *)((uint32_t)(dev[STD_IN].pcb.prev) -
(uint32_t)&(((ProcessTable *)0)->blocked));
dev[STD_IN].pcb.prev = (dev[STD_IN].pcb.prev)->prev;
(dev[STD_IN].pcb.prev)->next = &(dev[STD_IN].pcb);

pt->state = STATE_RUNNABLE;
pt->sleepTime = 0;
```

3.2 信号量

3.2.1 `syscallSemInit`

参数为`value`，即希望初始化时赋予`Semaphore`成员`value`的值，此函数比较简单，首先在`sem`中找到一块`state`为`0`的信号量，即未被使用过的，并将其`state`设置为1并将`value`设置为传入的参数`value`，最后初始化`sem`中的双向链表。

3.2.2 `syscallSemWait`

相当于`P`操作，将`value`自减，如果`value`小于0，则阻塞当前进程，方法与3.1中相同，不过`dev`变成了`sem`。

```
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);

pcb[current].state = STATE_BLOCKED;
asm volatile("int $0x20");
```

3.2.3 `syscallSemPost`

相当于`V`操作，将`value`自增，如果`value`小于等于0，则释放一个进程。

```
ProcessTable *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);

pt->state = STATE_RUNNABLE;
pt->sleepTime = 0;
```

3.2.4 `syscallSemDestroy`

销毁信号量

```
sem[i].state = 0;
asm volatile("int $0x20");
```

3.3 哲学家就餐问题

在`app/main.c`中定义`sem_t forks[5]`用来表示5个叉子

并定义两个函数

```
void getforks(int i);
void putdownforks(int i);
```

其中的参数`i`用于表示第几个哲学家（0-4）

```
void getforks(int i) {
    if (i % 2 == 0) {
        sem_wait(forks + i);
```

```
        sem_wait(forks + ((i + 1) % 5));
    }
    else {
        sem_wait(forks + ((i + 1) % 5));
        sem_wait(forks + i);
    }
}

void putdownforks(int i) {
    sem_post(forks + i);
    sem_post(forks + ((i + 1) % 5));
}
```

拿起叉子前，需要先对左手（右手）和右手（左手）边的叉子进行查询（分为奇数和偶数），即`P`操作，拿到两个叉子后，便可以`eat`，之后放下叉子，即`V`操作。

在`main`函数中，首先创建5个子进程，`pid`为2，3，4，5，6，

```
int start = getpid() - 2;
if (start >= 0) {
    while (1) {
        printf("Philosopher %d: think\n", start + 1);
        sleep(128);
        getforks(start);
        printf("Philosopher %d: eat\n", start + 1);
        sleep(128);
        putdownforks(start);
    }
}
```

`start`用于表示哲学家编号，由于之前存在内核进程和父进程，故需要`getpid()-2`

注意：由于需要创建5个子进程，需要修改`memory.h`中定义的`NR_SEGMENTS`以及`MAX_SEM_NUM`

在`irqhandle.c`中的`syscallWriteStdOut`中加入`putchar()`，并将终端打印结果重定向到`output`，实验结果如图所示。

以上是本次实验报告的全部内容，感谢阅读！

.