南京大学本科生实验报告

课程名称： **操作系统**

学院	计算机科学与技术		
学号	191220029	姓名	傅小龙
Email	1830970417@qq.com		

1.实验名称

lab4-进程同步

2.实验进度

我完成了所有内容。

3. 实验结果

3.1 实现格式化输入函数 & 3.2 实现信号量相关系统调用

```
njucs@njucs-VirtualBox:~/OSlab/lab4/lab4-191220029/lab4$ make play
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 2021 0xabc
```

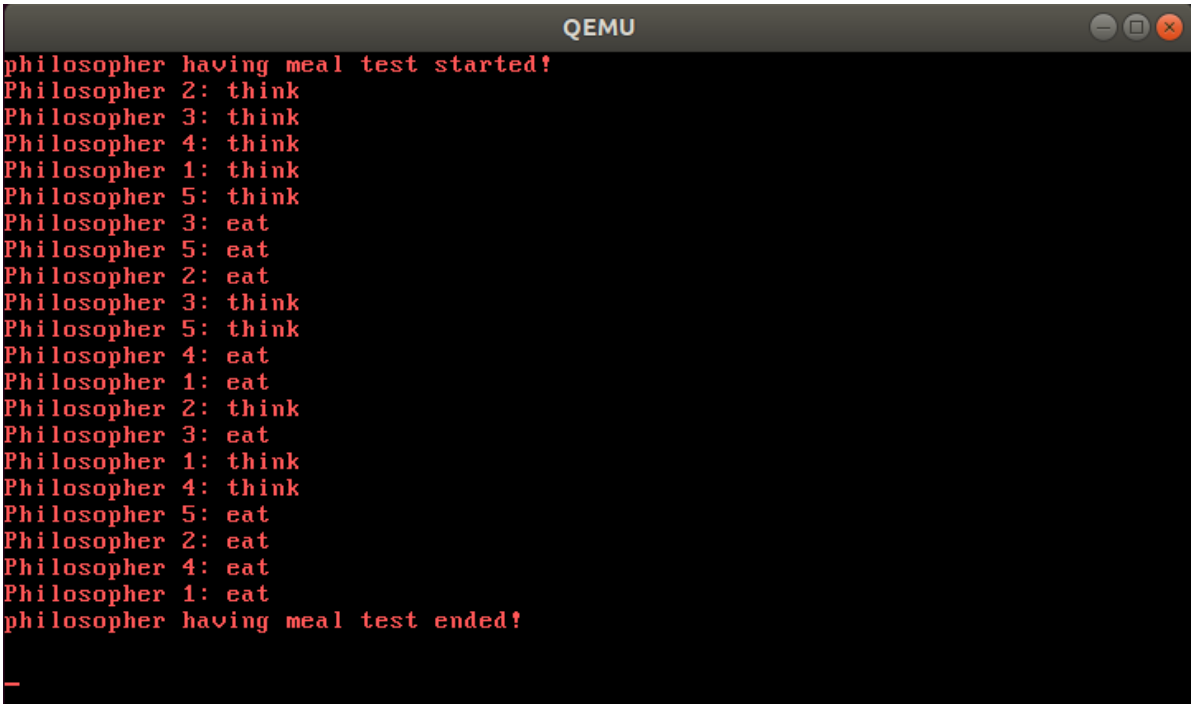
QEMU

```
Input:" Test %c Test %6s %d %x"
Test a Test oslab 2021 0xabc
Ret: 4; a, oslab, 2021, abc.
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.
```

3.3 基于信号量解决进程同步问题

3.3.1 哲学家就餐问题

每位哲学家的用餐次数限制由`./app/main.c`中的宏`EAT_TIME`给出. 哲学家在完成`EAT_TIME`次数的用餐后将退出. 运行结果如下图(`EAT_TIME = 2`):



3.3.2 生产者-消费者问题

父进程扮演消费者角色，4个子进程扮演生产者角色。生产者产出的产品存放在`product[SIZE_OF_BUFFER]`中，每个生产者生产`NUM_TO_PRODUCE`个产品后退出，父进程在消费完`4 * NUM_TO_PRODUCE`个产品后退出. 运行结果如下图：(`SIZE_OF_BUFFER = 5, NUM_TO_PRODUCE = 2`)

3.3.3 读者-写者问题

每个读者读`ReadTime`次后退出，每个写者写`WriteTime`次后退出. 运行结果如下：(`ReadTime = 2, WriteTime = 3`)

```
Reader and Writer test started!
Writer 1: write
Writer 5: write
Writer 6: write
Reader 2: read, total 1 reader
Reader 3: read, total 2 reader
Reader 4: read, total 3 reader
Reader 2: read, total 2 reader
Reader 3: read, total 2 reader
Reader 4: read, total 2 reader
Writer 1: write
Writer 5: write
Writer 6: write
Writer 1: write
Writer 5: write
Writer 6: write
Reader and Writer test ended!
```

4.代码修改处

`lib\lib.h`：添加`SYS_PID`宏定义，添加`pid_t getpid()`的声明.

`lib\syscall.c`:添加`pid_t getpid()`系统调用

`app\main.c`添加哲学家就餐问题、生产者-消费者问题、读者-写者问题的实现

`kernel\kernel\irqHandle`:

添加`SYS_PID`宏定义

实现`void syscallSemWait(...)`,`void syscallSemInit(...)`,`void syscallSemPost`,`void syscallSemDestroy(...)`,`void syscallReadStdIn(...)`,`void keyboardHandle(...)`

`kernel\include\x86\memory.h`:

修改`MAX_SEM_NUM`值为5（否则可用的信号量数量不够）

修改`NR_SEGMENTS`值为14 （否则无法支持同时运行5个线程）

5. 实验中遇到的问题及思考

生产者-消费者问题中产品的缓存队列in指针被生产者进程共享，读者-写者问题中的读者数也是共享变量。以读者-写者问题为例：

读者-写者问题中，读者读数据时要打印的信息中，活跃的读者数量为各读者进程所共享的变量，因而需要一块共享内存.

首先想到了在父进程的空间定义这一变量`Rcount`，初始化为0，然后作为参数传给读者进程所要执行的函数：

```
int Rcount = 0;
for(int i = 0; i < 3; i++){
    ret = fork();
    if(ret == 0){
        Reader(i, &RCountMutex, &WriteMutex, Rcount);
        exit(0);
    }
}
```

然后在`Reader(...)`函数内直接对`Rcount`操作。在实际运行中发现，某一读者进程对`Rcount`增1，但在其他读进程发现`Rcount`值仍为初值0.

这是由于访问的是子进程空间内的`Rcount`，并不是父进程空间的`Rcount`. 故需要将父进程的数据段选择子取出，在子进程内通过指定父进程的数据段选择子的方式访问`Rcount`.

```
int sel = 0;
asm volatile("movw %%ds, %0":"=r"(sel));
...//create reader process
    Reader(i, &RCountMutex, &WriteMutex, Rcount, sel);
```

但是这么写还是不行。由于传参采用形参，`Rcount`将被压入栈，实际传给子进程的是`Rcount`的值，而非是`Rcount`的地址，故需要将函数中的`Rcount`改为指针形式传入：

```
Reader(i, &RCountMutex, &WriteMutex, &Rcount, sel);
```

这样，在读者进程内就可以通过如下方式访问并修改`Rcount`的值：

```
asm volatile("movw %0, %%es"::"m"(fsel));
asm volatile("movl %%es:(%0), %1"::"r"(Rcount), "r"(*Rcount));
.../* *Rcount -= 1; OR *Rcount += 1; */
asm volatile("movl %0, %%es:(%1)"::"r"(*Rcount), "r"(Rcount));
```