

# 操作系统Lab4报告

---

## 基本信息

---

姓名：余帅杰

学号：181860077

邮箱：[3121416933@qq.com](mailto:3121416933@qq.com)

## 实验进度

---

### 完成的工作

---

完成了所有的工作

成功的实现了scanf

成功的实现了共享内存

成功实现了Sem一系列的处理例程

成功的解决了多个进程同步问题

### 修改位置

---

修改了irqHandle里的函数

1. keyboardHandle
2. syscallWriteShMem
3. syscallReadStdIn
4. syscallReadShMem
5. syscallSemInit
6. syscallSemWait
7. syscallSemPost
8. syscallSemDestroy

修改了main函数，以及各个进程同步问题的main函数

## 实验思路

---

### 完成Scanf任务

---

充分的阅读框架代码之后简单的来分析一下处理过程

#### scanf函数

进入scanf之后，主循环的结束于format的遍历结束（也就是scanf的参数）

在每一次外层遍历的时候都会对buffer进行判断，如果buffer为空，就会唤起readstdin

## readStdin

readstdin的任务就是把keyboard buffer里的内容进行提取，返回值是提取的数目（或者是阻塞时返回-1），反复提取直到触发上述的情况。同时readStdin会修改value使得当前的进程独占输入数据（其他的进程触发阻塞返回-1）。

## Keyboard

对于KeyBoard的任务就是在每一次按键的时候，中断被唤起往KeyHuffer里写数据。

所以可以简单的认为Keyboard函数有一点点独立开，两部分是基于buffer进行交流的

## 概述

那么任务和调试方法等就很明确了

keyboard负责被叫起来的时候把键码放进buffer，然后调整tail指针，如果有卡死的进程负责唤醒

readStdin就是负责把value减小独占输入数据，然后移动head指针一直到tail保证读取完键盘输入

scanf就是遍历输入参数，然后发现可用数据buffer空了就去中断唤起readStdin去再拿数据

对于测试样例就是，一开始scanf就发现了可用数据buffer为空，然后readStdin被调用并被挂起，随后键盘按下唤醒readStdin，然后读取这个键码并返回，scanf获取数据后开始处理数据，直到处理完又回到刚才的状态。

实现起来就简单了

## shaMem任务

---

这个任务实际上就是对一块数组的读写，实际上这种任务已经做过很多次了（readElf等）

唯一需要注意的就是需要试用内联汇编来处理拷贝的任务，而不能直接的赋值

其他的各个参数手册里都介绍了，看不懂的通过测试样例也可以猜到意思

## Sem系列的处理例程

---

### init

实际上就是找一块可以用的，然后把信号值赋值，类似于pcb找一块可以用的任务

### Wait

实际上就是P操作，减少信号值，考虑是否要阻塞当前进程，并放入等待链表

阻塞和放入链表参考手册，最后唤起时钟中断做进程重选

### Post

post是相对的V操作，用于还信号值，相对的就是考虑从链表里那一个出来运行

### Destory

销毁信号量，修改state参数

# 进程同步问题的解决

---

上课讲过，同时手册里详细的伪码可以参考

这一部分较简单

需要注意的就是Fork实际上创建的是当时的内存拷贝，在处理读者-写者问题的时候，多个写者共享的Rcount需要注意不能用简单的变量进行共享（由于Fork的时候每一个子进程都自己拷贝了一份），所以需要调用之前的进程共享内存的实现

详细的分析在实验结果一部分进行展示

## 随机数生成

---

考虑用素数和余数计算随机数

设置一个seed，随后的每一次的使用更新seed

```
1  int seed=39;
2  int myrand()
3  {
4      int a=114514;
5      int c=13277;
6      int d=73;
7      seed=(a*seed+c)%d;
8      return seed;
9  }
10
```

## 实验结果

---

### scanf

---

可以看到下图的输入是完全正常的

```
njucs@njucs-VirtualBox: ~/workspace/lab4-STUID/lab4
File Edit View Search Terminal Help
Name: ., Inode: 2, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: initrd, Inode: 3, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13608.
LS success.
1016 inodes and 3887 data blocks available.
ls /usr
Name: ., Inode: 4, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: print, Inode: 5, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13604.
Name: bounded_buffer, Inode: 6, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13740.
Name: philosopher, Inode: 7, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13712.
Name: reader_writer, Inode: 8, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13652.
LS success.
1016 inodes and 3887 data blocks available.
cat bootloader/bootloader.bin kernel/kMain.elf fs.bin > os.img
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 operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.
Test a Test oslab 2020 0xabc
█

QEMU
Input:" Test %c Test %6s %d %x"
Ret: 4: a, oslab, 2020, abc.
Input:" Test %c Test %6s %d %x"
```

## ShaMem

测试原理就是子进程不断的修改值，然后写入共享内存，然后父进程去读取内存

```
Process: %d %d\n" data data1);
njucs@njucs-VirtualBox: ~/workspace/lab4-STUID/lab4
File Edit View Search Terminal Help
ls /boot
Name: ., Inode: 2, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: initrd, Inode: 3, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13608.
LS success.
1016 inodes and 3887 data blocks available.
ls /usr
Name: ., Inode: 4, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: ., Inode: 5, Type: 1, LinkCount: 1, BlockCount: 14, Size: 13608.

QEMU
Father Process: 2020, 0
Child Process: 2020, 1000
Father Process: 2020, 2020
Child Process: 3020, 1000
Father Process: 2020, 3020
Child Process: 4020, 1000
Father Process: 2020, 4020
Child Process: 5020, 1000
-
```

## Sem系列

测试原理基本是父子进程取信号量，子进程还

```
QEMU
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.
-
```

## 进程通信问题

### 生产者消费者

参数设置，四个生产者一个消费者，buffer大小为3（可以体现bufferfull）

取一段为例分析正确性如下

先执行Consumer，wait fullbuffer

然后执行了4个producer，三个producer抢到了emptybuffer（buffer大小，也就是初始化emptybuffer），随后3得到了mutex，生产，归还mutex，同理4，5进行生产，生产之后释放了mutex和fullbuffer，consumer开始处理。处理之后唤起emptybuffer，之前被挂起排队的6开始生产。之后就同理了。



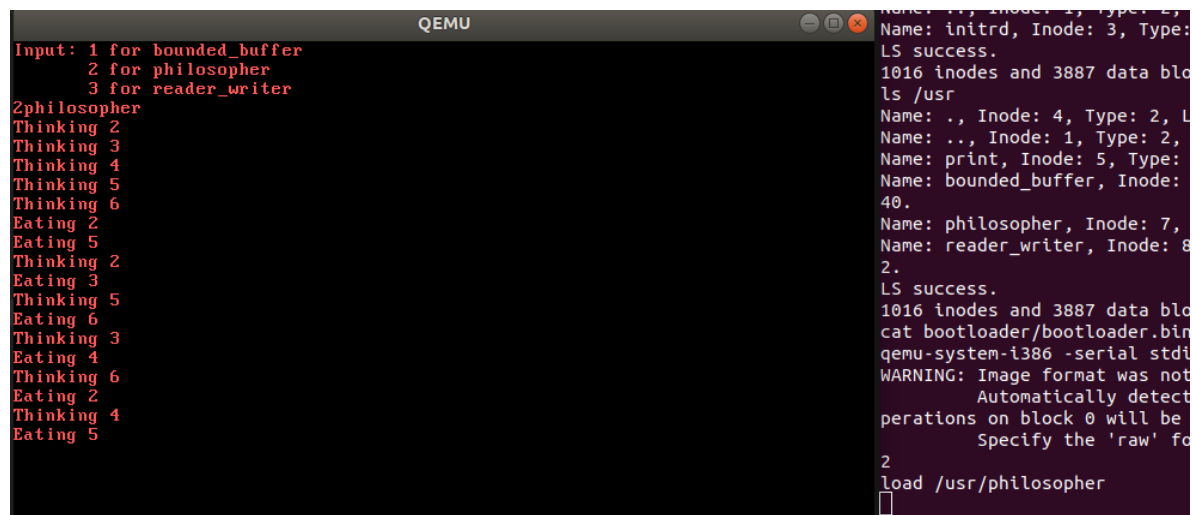
```
QEMU
Input: 1 for bounded_buffer
      2 for philosopher
      3 for reader_writer
1bounded_buffer
Producer 3: produce
Producer 4: produce
Producer 5: produce
Consumer : consume
Producer 6: produce
Consumer : consume
Producer 3: produce
Consumer : consume
Producer 4: produce
Consumer : consume
Producer 5: produce
```

## 哲学家

哲学家问题可以看到一开始大家都思考

然后2和5抢到叉子进餐，符合题设

随后规划叉子进入思考，进餐者各自移动了一位，显然依旧符合题设随后循环。



```
QEMU
Input: 1 for bounded_buffer
      2 for philosopher
      3 for reader_writer
2philosopher
Thinking 2
Thinking 3
Thinking 4
Thinking 5
Thinking 6
Eating 2
Eating 5
Thinking 2
Eating 3
Thinking 5
Eating 6
Thinking 3
Eating 4
Thinking 6
Eating 2
Thinking 4
Eating 5
```

```
Name: initrd, Inode: 3, Type: 2,
LS success.
1016 inodes and 3887 data blocks
ls /usr
Name: ., Inode: 4, Type: 2, L
Name: .., Inode: 1, Type: 2,
Name: print, Inode: 5, Type:
Name: bounded_buffer, Inode:
40.
Name: philosopher, Inode: 7,
Name: reader_writer, Inode: 8
2.
LS success.
1016 inodes and 3887 data blocks
cat bootloader/bootloader.bin
qemu-system-i386 -serial stdio
WARNING: Image format was not
Automatically detect
operations on block 0 will be
Specify the 'raw' fo
2
load /usr/philosopher
```

# 读者写者问题

先是执行读者，读者先抢占Count的控制权，随后sleep卡住

执行写者，直接拿到了WriteMutex，抢到Count控制权刚sleep结束的reader排在队列的最后面

三个写者写完了之后释放了WriteMutex（重新再来的时候就得排在等候队列的最后面了），这个时候排在最后读者醒了，读取，这个和描述的互斥访问一致。随后读者里还有的由于Count阻塞的被唤醒读取，这个时候有读者再读，不需要抢占写者的控制权，直接一起读就可以了。

最后归还，排队的写者又醒过来了

```
sem_init(&CountMutex,1);
int Rcount=0;
write(SH_MEM, (uint8_t *)&Rcount, 4, 0);
for (int i = 0; i < 6; ++i)
{
    if (fork() == 0)
    {
        //int pid = getpid();
        //printf("%d %d\n",pid,i);
        if(i<3)
        {
            Reader(i,&WriteMutex,&CountMutex);
        }
        else
        {
            Writer(i,&WriteMutex);
        }
    }
}
exit();
return 0;
}
```

**QEMU**  
Input: 1 for bounded\_buffer  
2 for philosopher  
3 for reader\_writer  
3reader\_writer  
Write 5  
Write 6  
Write 7  
Read 2 total 1  
Read 3 total 2  
Read 4 total 3  
Write 5  
Write 6  
Write 7  
Read 2 total 1  
Read 3 total 2  
Read 4 total 3  
Write 5  
—

# 其他

主要是学习到了信号量相关操作的实现

同时实际上手处理进程同步的问题

中途也遇到了很多问题，为了解决问题也充分的阅读了框架代码，了解了scanf的实现原理以及其他相关的部分代码

