

OS Lab4

学号: 181860154

姓名: 朱倩

邮箱: infinite0124@163.com

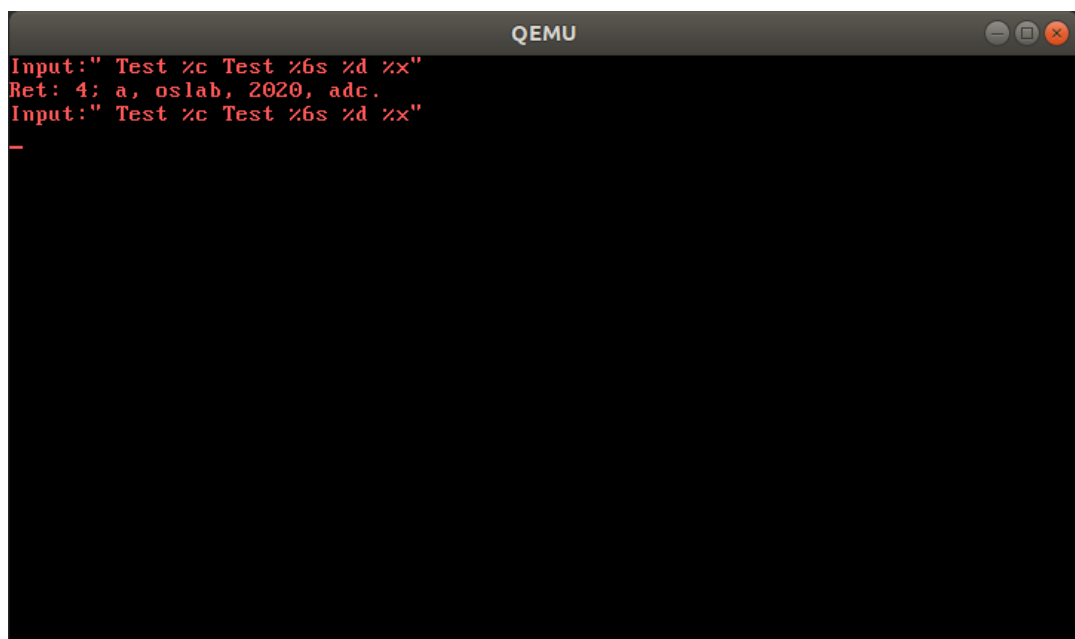
一、实验进度

完成了所有必做内容

二、实验结果

1. 实现格式化输入函数

在app/main.c中输入测试代码后make play, 输入 `Test a Test oslab 2020 0xadc` 后的输出结果:



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

2. 实现进程通信

在app/main.c中输入测试代码后make play的输出结果:

```
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
```

3. 实现信号量

在app/main.c中输入测试代码后make play的输出结果:

```
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.
```

4. 解决进程同步问题

1) 生产者-消费者问题

```
QEMU
Input: 1 for bounded_buffer
       2 for philosopher
       3 for reader_writer
bounded_buffer
Producer 0: produce
Producer 1: produce
Producer 2: produce
Producer 3: produce
Consumer : consume
Producer 0: produce
Producer 1: produce
Consumer : consume
Producer 2: produce
Consumer : consume
Producer 3: produce
Consumer : consume
Producer 0: produce
Consumer : consume
Producer 1: produce
Consumer : consume
Producer 2: produce
Consumer : consume
Producer 3: produce
Consumer : consume
```

缓冲区大小设置为5，4个生产者，1个消费者。

从输出结果来看，缓冲区中产品数量变化为1,2,3,4,3,4,5,4,5,4,5...一直没有超过缓冲区大小。

2) 哲学家就餐问题

```
QEMU
Input: 1 for bounded_buffer
       2 for philosopher
       3 for reader_writer
philosopher
Philosopher 1: think
Philosopher 3: think
Philosopher 0: think
Philosopher 2: think
Philosopher 4: think
Philosopher 1: eat
Philosopher 3: eat
Philosopher 1: think
Philosopher 0: eat
Philosopher 3: think
Philosopher 2: eat
Philosopher 0: think
Philosopher 4: eat
Philosopher 2: think
Philosopher 1: eat
Philosopher 4: think
Philosopher 3: eat
Philosopher 1: think
Philosopher 0: eat
```

哲学家就餐问题中，要保证座位相邻的哲学家不同时用餐。哲学家有两种状态，think时放下叉子，eat时拿起左右两支叉子。

从输出结果来看，发生的事件为：（下面用数字代表哲学家或叉子编号）

哲学家1,3,0,2,4都在思考；

哲学家1拿到叉子1,2开始吃；

哲学家3拿到叉子3,4开始吃；

哲学家1放下叉子1,2开始思考；

哲学家0拿到叉子0,1开始吃;

哲学家3放下叉子3,4开始思考;

哲学家2拿到叉子2,3开始吃;

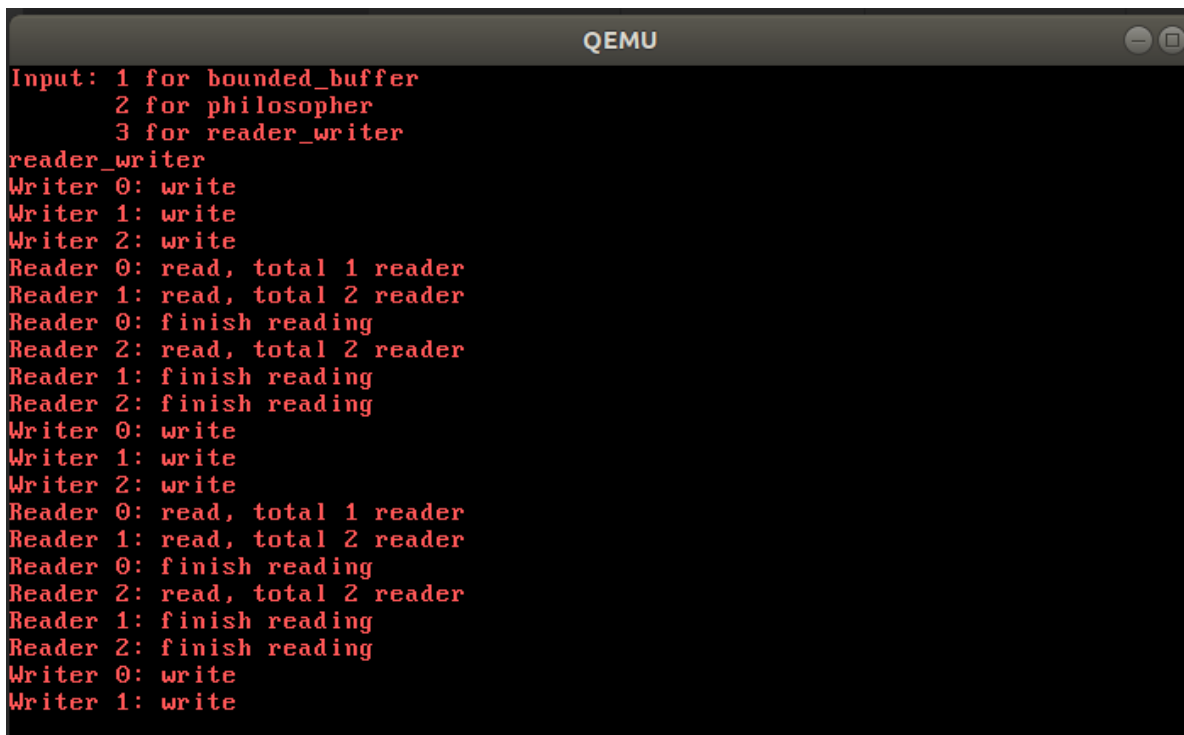
哲学家0放下叉子0,1开始思考;

哲学家4拿到叉子4,0开始吃;

.....

各哲学家的think和eat动作穿插进行，每位哲学家都有吃的机会，没有叉子被两位哲学家同时使用。

3) 读者-写者问题



```
QEMU
Input: 1 for bounded_buffer
       2 for philosopher
       3 for reader_writer
reader_writer
Writer 0: write
Writer 1: write
Writer 2: write
Reader 0: read, total 1 reader
Reader 1: read, total 2 reader
Reader 0: finish reading
Reader 2: read, total 2 reader
Reader 1: finish reading
Reader 2: finish reading
Writer 0: write
Writer 1: write
Writer 2: write
Reader 0: read, total 1 reader
Reader 1: read, total 2 reader
Reader 0: finish reading
Reader 2: read, total 2 reader
Reader 1: finish reading
Reader 2: finish reading
Writer 0: write
Writer 1: write
```

3个读者，3个写者，读写、写写互斥，读读不互斥。

为了让进行过程更清晰，增加了读者完成阅读的输出。（Rcount--时输出）

从输出结果来看，写者写的时候没有读者在阅读；

读者的计数也跟当前输出read，未输出finish reading的读者数量相符。

三、实验修改的代码位置

1. 实现格式化输入函数：

lab4/kernel/kernel/irqHandle.c:

```
void keyboardHandle(struct TrapFrame *tf);
/*将读取到的keyCode放入到keyBuffer中,唤醒阻塞在dev[STD_IN]上的一个进程*/

void syscallReadStdIn(struct TrapFrame *tf);
/*如果dev[STD_IN].value == 0 , 将当前进程阻塞在dev[STD_IN]上;
进程被唤醒, 读keyBuffer 中的所有数据*/
```

2. 实现进程通信:

lab4/kernel/kernel/irqHandle.c:

```
void syscallReadShMem(struct TrapFrame *tf); //读共享内存中的数据
void syscallWriteShMem(struct TrapFrame *tf); //写共享内存中的数据
```

3. 实现信号量:

lab4/kernel/kernel/irqHandle.c:

```
void syscallSemInit(struct TrapFrame *tf); //信号量的初始化
void syscallSemWait(struct TrapFrame *tf); //相当于P操作
void syscallSemPost(struct TrapFrame *tf); //相当于V操作
void syscallSemDestroy(struct TrapFrame *tf); //信号量的销毁
```

4. 解决进程同步问题:

1) 生产者-消费者问题

lab4/bounded_buffer/main.c:

```
int main(void);
/*fork出5个子进程, 4个调用生产者处理函数producer并传入id, 1个调用消费者处理函数
consumer*/

void producer(int id, sem_t *empty, sem_t *mutex, sem_t *full);
/*生产者处理函数, id表示生产者编号, empty, mutex, full为用到的信号量*/

void consumer(sem_t *empty, sem_t *mutex, sem_t *full);
/*消费者处理函数, empty, mutex, full为用到的信号量*/
```

2) 哲学家就餐问题

lab4/pilosopher/main.c:

```
int main(void);
/*fork出5个子进程, 每个子进程调用哲学家处理函数philosopher, 并传入id*/

void philosopher(int i);
/*哲学家处理函数, i表示哲学家编号*/

sem_t forks[5];
/*信号量fork[i]作为全局变量使用*/
```

3) 读者-写者问题

lab4/reader_writer/main.c:

```
int main(void);  
/*fork出10个子进程，5个调用读者处理函数reader并传入id，5个调用写者处理函数writer并  
传入id*/  
  
void reader(int id,sem_t *CountMutex,sem_t *WriteMutex);  
/*读者处理函数，id表示读者编号，CountMutex和WriteMutex为用到的信号量*/  
  
void writer(int id,sem_t *write_mutex);  
/*写者处理函数，id表示写者编号，writeMutex为用到的信号量*/
```