操作系统lab4【齐心协力-进程通信】

201220199 肖丹妮

- 4个exercise
- 3个task

Exercises

exercise1

如果5位哲学家已同时拿起左边（或右边）的叉子，当每位哲学家再次试图拿起右边（或左边）的叉子时，会出现死锁。

exercise2

任意时刻只能至多允许一位哲学家拿到两把叉子吃通心面，并发性不高。

exercise3

申请信号量mutex，用于控制至多允许4位哲学家同时吃通心面，初值为4

```
#define N 5                     // 哲学家个数
semaphore fork[5];              // 信号量初值为1
semaphore mutex;                // 互斥信号量，初值4
void philosopher(int i){        // 哲学家编号：0-4
  while(TRUE){
    think();                    // 哲学家在思考
    P(mutex);                   // 进入临界区
    P(fork[i]);                 // 去拿左边的叉子
    P(fork[(i+1)%N]);           // 去拿右边的叉子
    eat();                      // 吃面条
    V(fork[i]);                 // 放下左边的叉子
    V(fork[(i+1)%N]);           // 放下右边的叉子
    V(mutex);                   // 退出临界区
  }
}
```

exercise4

fullBuffers、emptyBuffers用于控制生产者和消费者的同步问题，emptyBuffers指示能否向缓冲区中放入产品，fullBuffers指示能否从缓冲区中取出产品。

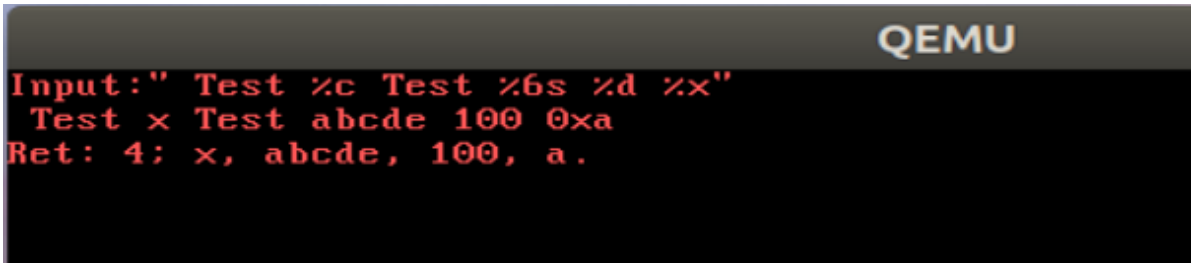
两者的直观含义：

- emptyBuffers：可用的空缓冲区个数
- fullBuffers：缓冲区中可取的产品个数

Tasks

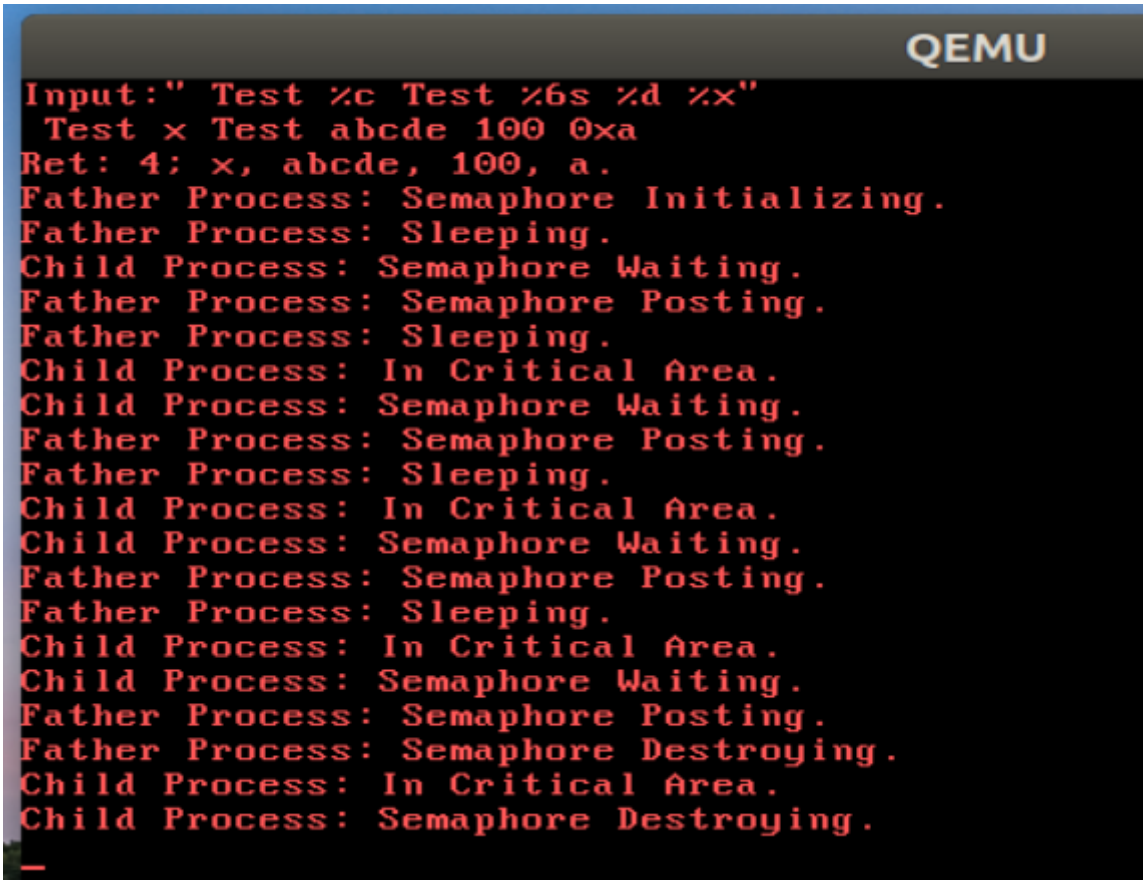
task1

测试scanf的结果如下：



task2

信号量测试结果如下：



task3

哲学家就餐问题

- 代码：

实现了系统调用getpid

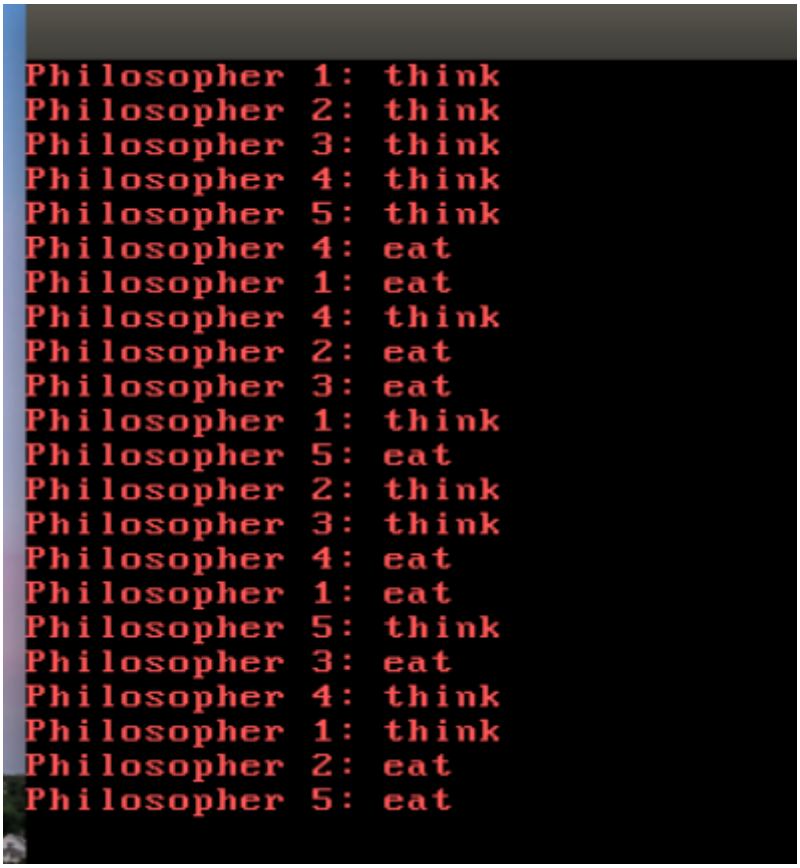
```
#define SYS_PID 7
```

```
int getpid(){
    return syscall(SYS_PID, 0, 0, 0, 0, 0);
}
```

```
case SYS_PID:
        syscallPid(sf);
        break; // for SYS_PID
```

```
void syscallPid(struct StackFrame *sf)
{
    sf->eax = pcb[current].pid;
    return;
}
```

- 运行截图如下：



生产者-消费者问题

- 代码：

在irqHandle.c中声明以下数据，其中buffer等变量在kvm.c中定义

```
#define BUF_APPEND 8
#define BUF_TAKE 9
#define bufN 10
extern int buffer[bufN];
extern int product;
extern int in;
extern int out;
```

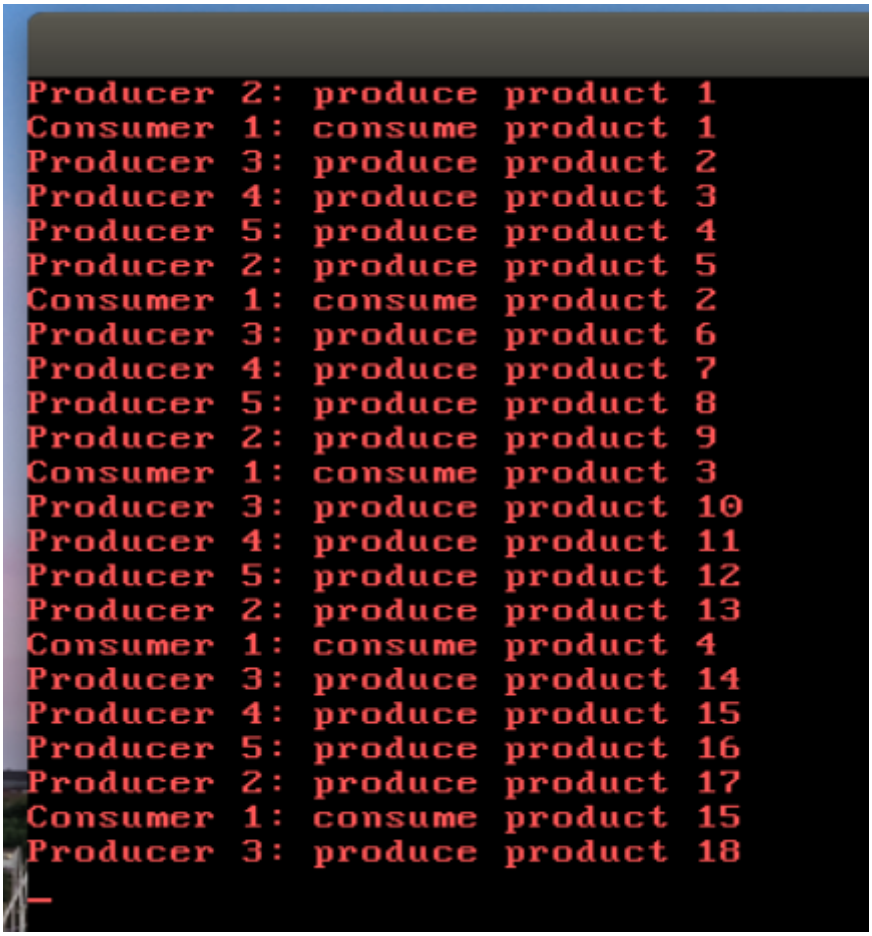
实现了系统调用append_to_buffer以及take_from_buffer，返回值分别是生产的产品和取出的产品

```
int append_to_buffer(){
    return syscall(BUF_APPEND, 0, 0, 0, 0, 0);
}

int take_from_buffer(){
    return syscall(BUF_TAKE, 0, 0, 0, 0, 0);
}
```

```
case BUF_APPEND:
        syscallBufAppend(sf);
        break; // for BUF_APPEND
    case BUF_TAKE:
        syscallBufTake(sf);
        break; // for BUF_TAKE
```

```
void syscallBufAppend(struct StackFrame *sf){
    product++;
    buffer[in] = product;
    in = (in+1)%bufN;
    sf->eax = product;
    return;
}

void syscallBufTake(struct StackFrame *sf){
    sf->eax = buffer[out];
    out = (out+1)%bufN;
    return;
}
```

- 运行截图如下：


Producer 2: produce product 1
Consumer 1: consume product 1
Producer 3: produce product 2
Producer 4: produce product 3
Producer 5: produce product 4
Producer 2: produce product 5
Consumer 1: consume product 2
Producer 3: produce product 6
Producer 4: produce product 7
Producer 5: produce product 8
Producer 2: produce product 9
Consumer 1: consume product 3
Producer 3: produce product 10
Producer 4: produce product 11
Producer 5: produce product 12
Producer 2: produce product 13
Consumer 1: consume product 4
Producer 3: produce product 14
Producer 4: produce product 15
Producer 5: produce product 16
Producer 2: produce product 17
Consumer 1: consume product 15
Producer 3: produce product 18