

《操作系统》实验四报告

201220069 周心同

一、实验目的

进程间通信的方式多种多样，常见的有管道，信号量，信号，共享内存，套接字.....

本次实验会让大家体会一下信号量PV操作是如何进行的。

二、实验内容

- 本次实验共有如下任务点需要完成（编号都是从1开始计数）：

- 4个exercise
- 3个task

- 完成内容：

- 4个exercise
- 3个task（包括选做）

三、实验过程和结果（按照手册顺序）

exercise1：请回答一下，什么情况下会出现死锁。

当每个哲学家都同时拿了左手边的叉子时，此时桌子上没有一个叉子，同时每个哲学家都在等待右手边的叉子，产生死锁现象。

exercise2：说一下该方案有什么不足？（答出一点即可）

最多只能有一个哲学家在吃面条，叉子的利用效率低。

exercise3: 正确且高效的解法有很多，请你利用信号量PV操作设计一种正确且相对高效（比方案2高效）的哲学家吃饭算法。（其实网上一堆答案，主要是让大家多看看不同的实现。）

规定1，3号哲学家先拿左手边的叉子，0，2，4号哲学家先拿右手边的叉子，这样0，1号相互竞争，2，3号相互竞争，（当二者之一抢到叉子时，另一位会继续等待）而0，3，4号之间一定不会发生死锁现象（同样会等待，故能有先后顺序）。

这个方法有哲学家要进餐时，至少有一个哲学家能立马拿到叉子，更好的情况有两个哲学家能同时拿到叉子，显然效率比方案2高。

```
#define N 5                // 哲学家个数
semaphore fork[5];        // 信号量初值为1
void philosopher(int i){   // 哲学家编号：0-4
    while(TRUE){
        if(i%2 == 1)
        {
            think();        // 哲学家在思考
            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
        }
        else
        {
            think();        // 哲学家在思考
            P(fork[(i+1)%N]); // 去拿右边的叉子
            P(fork[i]);      // 去拿左边的叉子
            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
        }
    }
}
```

exercise4: 为什么要用两个信号量呢? emptyBuffers和fullBuffer分别有什么直观含义?

这不是三个信号量吗?

信号量mutex控制互斥访问, 防止有多个线程访问缓存区。

另外一对信号量emptyBuffers和fullBuffer控制条件同步, emptyBuffers表示缓存区有多少空位可以让生产者生产, fullBuffer表示缓存区有多少生产的东西可以让消费者取用。

每个信号量可以拒绝一种条件下的访问

故对于下面三种拒绝访问的情况, 必须要有三个信号量:

任何时刻只能有一个线程操作缓冲区 (互斥访问)

缓冲区空时, 消费者必须等待生产者 (条件同步)

缓冲区满时, 生产者必须等待消费者 (条件同步)

task1: 完成格式化输入

完成这一步后请测试scanf, 并在实验报告展示结果

这里keyboardhandle是把character而不是code放入buffer了吧, 手册有误。

弄了半天才发现Test前面有空格, 这测试太不友好了。

这里根据 dev[STD_IN].value 的值, 小于0说明有进程被阻塞, 返回-1, 为0则将该进程阻塞, 为1则唤醒该进程读取缓存区的字符并传到用户进程。

```
Input: " Test %c Test %6s %d %x"
Test a Test abc 123 0x12
Ret: 4; a, abc, 123, 12.
Father Process: Semaphore Initializ
```

task2: 实现下面四个函数

每个函数首先判断一下索引范围是否合法，部分要判断对应的state是否符合。

init初始化一下状态，destroy把state改为0，而wait和post分别对应P和V操作，结合信号量介绍和手册给出的代码就能写了。

```
Father Process: Semaphore Initializing.
Father Process: Sleeping.
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 Waiting.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Father Process: Semaphore Posting.
Father Process: Semaphore Destroying.
Child Process: In Critical Area.
Child Process: Semaphore Destroying.
```

task3: 完成app里面的下列问题，在报告里放上运行截图（注意在写其中一个问题时，把别的代码注释掉）。

哲学家就餐问题

我这里把think放在eat和V操作中间，表明eat完后think就是放下了叉子了。

另外子进程最多有四个，创建第五个就会失败，（进内核发现MAX_PCB_NUM为6，创建时0（内核进程）和1（用户进程）都被占据，故只能创建四个）所以有一个哲学家只能放在父进程里了。

创建好进程后while循环框架代码即可。

```
QEMU
Philosopher 1: eat
Philosopher 4: eat
Philosopher 1: think
Philosopher 4: think
Philosopher 2: eat
Philosopher 0: eat
Philosopher 2: think
Philosopher 0: think
Philosopher 4: eat
Philosopher 1: eat
Philosopher 4: think
Philosopher 1: think
Philosopher 3: eat
Philosopher 3: think
Philosopher 0: eat
Philosopher 0: think
Philosopher 2: eat
Philosopher 2: think
Philosopher 4: eat
Philosopher 1: eat
Philosopher 4: think
-
```

生产者-消费者问题

设定一开始buffer容量为5。

一个生产者作为父进程，四个消费者为子进程。

创建好进程后while循环框架代码即可。

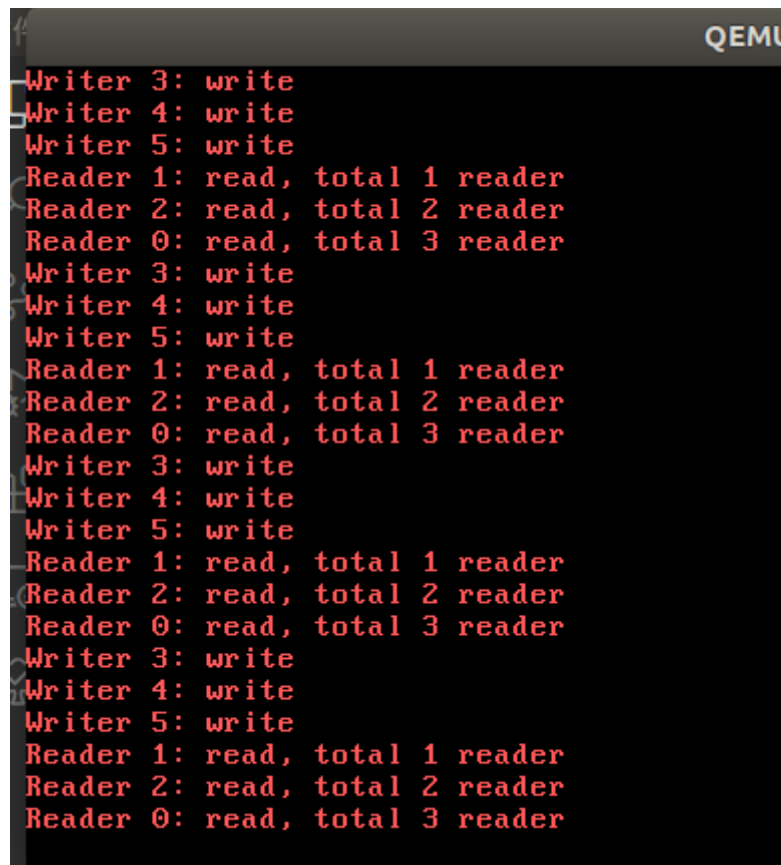
```
Producer 1: produce
Producer 2: produce
Producer 3: produce
Producer 4: produce
Consumer : consume
Producer 1: produce
Producer 2: produce
Consumer : consume
Producer 3: produce
Consumer : consume
Producer 4: produce
Consumer : consume
Producer 1: produce
Consumer : consume
Producer 2: produce
Consumer : consume
Producer 3: produce
Consumer : consume
Producer 4: produce
```

读者-写者问题

这里要求6个进程，不够，需要进内核改NR_SEGMENTS，这里改成17刚好够。

另外这里的Rcount必须要求能够进程间共享，直接在main里面创建发现不行，考虑到因为进程的id都可以从内核中获取，所以这里就把Rcount也设定在内核里从而获取（定义一个全局变量Rcount）。

除此之外对每个进程while循环框架代码即可。



```
Writer 3: write
Writer 4: write
Writer 5: write
Reader 1: read, total 1 reader
Reader 2: read, total 2 reader
Reader 0: read, total 3 reader
Writer 3: write
Writer 4: write
Writer 5: write
Reader 1: read, total 1 reader
Reader 2: read, total 2 reader
Reader 0: read, total 3 reader
Writer 3: write
Writer 4: write
Writer 5: write
Reader 1: read, total 1 reader
Reader 2: read, total 2 reader
Reader 0: read, total 3 reader
Writer 3: write
Writer 4: write
Writer 5: write
Reader 1: read, total 1 reader
Reader 2: read, total 2 reader
Reader 0: read, total 3 reader
```

四、实验的启示/意见和建议

这次实验总体比较简单，比如task3的三个问题前面代码全部都给出来了，就是debug浪费不少时间，比如读者写者问题弄半天才发现写Rcount应该是sf->ecx手打成了sf->ebx。

