1. **Concurrent Objects:**
   1. Please construct some linearizable execution histories for concurrent stacks.

example 1

s.push(2)

s.pop(1)

s.push(1)

s.pop(2)

example 2

s.push(2)

s.push(1)

s.pop(1)

s.pop(2)

* 1. **[optional]** Exercise 32 in Chapter 3.11

1.假设15行为线性化点：假设执行前队列为空：

A线程： q.enq(1)

B线程： q.enq(2) q.deq()

A,B线程并行执行，由于15行和16行之间执行存在时间间隔，执行的轨迹可能如下：

A先执行至第15行，取得位置0；

B执行至15行，取得位置1;

B执行第16行，item[1] = 2;

B执行deq操作，由于位置0现在仍为null,导致出队元素为位置1的2;

A执行16行，item[0] = 1;

这时，虽然A先执行到线性化点，但程序1的执行效果却是B先入队，在A还未执行完16行时，出队的自然是B线程入队的元素2。和在15行线性化结果不一致，说明15行不可能为线性化点。

2. 假设16行为线性化点:

q.enq(1) 15 16

A:

q.enq(2) 15 16

B:

在上图执行历史中，虽然A线程的线性化点在后面到达，但由于A先在15行获得较小的index，A线程的元素1仍进入队列的前面。说明程序的执行效果和在16行线性化不一致，

16行不可能为线性化点。

(3)没有线性化点不代表不可以线性化，指定可线性化点只是用来说明可线性化的一种方法，二者并不等价。分析可知enq满足线性化的形式化定义，说明enq是可线性化的。

1. **Monitors and Blocking Synchronizations:** Exercise 95 : (1) in Chapter 8.7

(1)用锁和条件实现储蓄账户对象:

**public** **class** Account {

**private** **int** balance;

Lock lock;

Condition condition;

**public** Account(**int** balance) {

**this**.balance = balance;

}

**public** **void** withdraw(**int** amount)**throws** InterruptedException{

lock.lock();

**try** {

**while**(balance<amount) {

condition.await();

}

balance -= amount;

} **finally** {

lock.unlock();

}

}

**public** **void** deposit(**int** amount) **throws** InterruptedException{

lock.lock();

**try** {

balance +=amount;

condition.notifyAll();

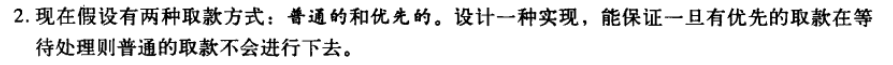
} **finally** {

lock.unlock();

}

}

}



给每个withdraw增加一个优先权标识，在执行withdraw之前进行判断，如果prior为HIGH则返回；获得锁后，将prior置为自己的优先权，释放锁之前再将prior置为LOW：

**public** **class** Account {

**private** **int** balance;

Lock lock;

Condition condition;

**enum** Prior { *LOW*, *HIGH*};

Prior prior = Prior.*LOW*;;

**public** Account(**int** balance) {

**this**.balance = balance;

}

**public** **void** withdraw(**int** amount, Prior prior) **throws** InterruptedException{

**if** ( prior == Prior.*LOW* && **this**.prior == Prior.*HIGH*) {

**return**;

}

lock.lock();

**try** {

**this**.prior = prior;

**while**(balance<amount) {

condition.await();

}

balance -= amount;

} **finally** {

**this**.prior = Prior.*LOW*;

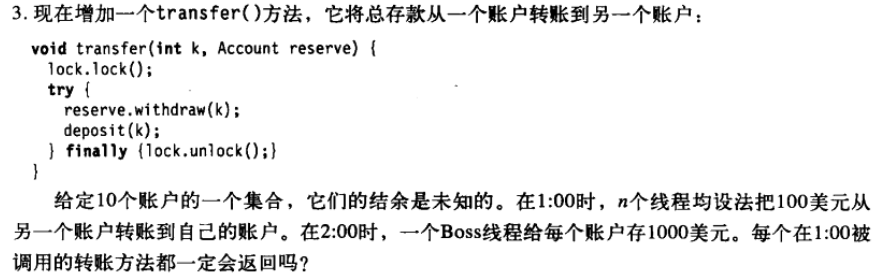
lock.unlock();

}

}

. . . . . .

}



会返回，最极端的情况是：某个账户余额不足100$，其他账户都要从其转帐，需要从一点等待到2点。但2点存入的1000$可以满足其他账户的withdraw操作，其他账户会依次获得锁，转账成功并返回。