26.1

首先查看loop.s

```
.main
.top
sub  $1,%dx
test $0,%dx
jgte .top
halt
```

可以看到该汇编程序将dx的值-1

然后与0进行比较 如果大于等于0就跳转回top标签

最后终止

所以如果dx初始值为0 那么dx将会从0变成-1

实际运行结果如下

```
-> % ./x86.py -p loop.s -t 1 -i 100 -R dx -c
ARG seed 0
ARG numthreads 1
ARG program loop.s
ARG interrupt frequency 100
ARG interrupt randomness False
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace
ARG regtrace dx
ARG cctrace False
ARG printstats False
ARG verbose False

   dx          Thread 0
    0
   -1   1000 sub  $1,%dx
   -1   1001 test $0,%dx
   -1   1002 jgte .top
   -1   1003 halt
```

26.2

由于指令中断设置为100

所以线程0不会被中断 直到所有指令 执行完毕 所以 dx由3到-1

线程0运行结束后运行线程1 同样的 dx由3到-1

运行结果如下：

```
-> % ./x86.py -p loop.s -t 2 -i 100 -a dx=3,dx=3 -R dx -c
```

```
ARG seed 0
ARG numthreads 2
ARG program loop.s
ARG interrupt frequency 100
ARG interrupt randomness False
ARG argv dx=3,dx=3
ARG load address 1000
ARG memsize 128
ARG memtrace
ARG regtrace dx
ARG cctrace False
ARG printstats False
ARG verbose False

   dx          Thread 0                Thread 1
    3
    2   1000 sub  $1,%dx
    2   1001 test $0,%dx
    2   1002 jgte .top
    1   1000 sub  $1,%dx
    1   1001 test $0,%dx
    1   1002 jgte .top
    0   1000 sub  $1,%dx
    0   1001 test $0,%dx
    0   1002 jgte .top
   -1   1000 sub  $1,%dx
   -1   1001 test $0,%dx
   -1   1002 jgte .top
   -1   1003 halt
    3   ----- Halt;Switch -----  ----- Halt;Switch -----
    2                            1000 sub  $1,%dx
    2                            1001 test $0,%dx
    2                            1002 jgte .top
    1                            1000 sub  $1,%dx
    1                            1001 test $0,%dx
    1                            1002 jgte .top
    0                            1000 sub  $1,%dx
    0                            1001 test $0,%dx
    0                            1002 jgte .top
   -1                            1000 sub  $1,%dx
   -1                            1001 test $0,%dx
   -1                            1002 jgte .top
   -1                            1003 halt
```

26.3

```
-> % ./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx -c -s 1
   dx          Thread 0                Thread 1
    3
    2   1000 sub  $1,%dx
    3   ------ Interrupt ------  ------ Interrupt ------
    2                            1000 sub  $1,%dx
    2                            1001 test $0,%dx
    2                            1002 jgte .top
    2   ------ Interrupt ------  ------ Interrupt ------
    2   1001 test $0,%dx
    2   1002 jgte .top
    1   1000 sub  $1,%dx
    2   ------ Interrupt ------  ------ Interrupt ------
    1                            1000 sub  $1,%dx
    1   ------ Interrupt ------  ------ Interrupt ------
    1   1001 test $0,%dx
    1   1002 jgte .top
    1   ------ Interrupt ------  ------ Interrupt ------
    1                            1001 test $0,%dx
    1                            1002 jgte .top
    1   ------ Interrupt ------  ------ Interrupt ------
    0   1000 sub  $1,%dx
    0   1001 test $0,%dx
    1   ------ Interrupt ------  ------ Interrupt ------
    0                            1000 sub  $1,%dx
    0                            1001 test $0,%dx
    0                            1002 jgte .top
    0   ------ Interrupt ------  ------ Interrupt ------
    0   1002 jgte .top
    0   ------ Interrupt ------  ------ Interrupt ------
   -1                            1000 sub  $1,%dx
    0   ------ Interrupt ------  ------ Interrupt ------
   -1   1000 sub  $1,%dx
   -1   1001 test $0,%dx
   -1   1002 jgte .top
   -1   ------ Interrupt ------  ------ Interrupt ------
   -1                            1001 test $0,%dx
   -1                            1002 jgte .top
   -1   ------ Interrupt ------  ------ Interrupt ------
   -1   1003 halt
   -1   ----- Halt;Switch -----  ----- Halt;Switch -----
   -1                            1003 halt
-> % ./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx -c -s 2
   dx          Thread 0                Thread 1
    3
    2   1000 sub  $1,%dx
    2   1001 test $0,%dx
    2   1002 jgte .top
    3   ------ Interrupt ------  ------ Interrupt ------
    2                            1000 sub  $1,%dx
    2                            1001 test $0,%dx
    2                            1002 jgte .top
    2   ------ Interrupt ------  ------ Interrupt ------
    1   1000 sub  $1,%dx
    2   ------ Interrupt ------  ------ Interrupt ------
    1                            1000 sub  $1,%dx
    1   ------ Interrupt ------  ------ Interrupt ------
    1   1001 test $0,%dx
    1   1002 jgte .top
    0   1000 sub  $1,%dx
    1   ------ Interrupt ------  ------ Interrupt ------
    1                            1001 test $0,%dx
    1                            1002 jgte .top
    0                            1000 sub  $1,%dx
    0   ------ Interrupt ------  ------ Interrupt ------
    0   1001 test $0,%dx
```

```
     0   1002 jgte .top
    -1   1000 sub  $1,%dx
     0   ------ Interrupt ------  ------ Interrupt ------
     0                            1001 test $0,%dx
    -1   ------ Interrupt ------  ------ Interrupt ------
    -1   1001 test $0,%dx
    -1   1002 jgte .top
     0   ------ Interrupt ------  ------ Interrupt ------
     0                            1002 jgte .top
    -1                            1000 sub  $1,%dx
    -1   ------ Interrupt ------  ------ Interrupt ------
    -1   1003 halt
    -1   ----- Halt;Switch -----  ----- Halt;Switch -----
    -1                            1001 test $0,%dx
    -1   ------ Interrupt ------  ------ Interrupt ------
    -1                            1002 jgte .top
    -1   ------ Interrupt ------  ------ Interrupt ------
    -1                            1003 halt
```

通过使用不同的随机数种子 可以看到两个线程的中断行为有很大差异

说明中断频率会改变这个程序的行为。

26.4

首先分析looping-race-nolock.s

```
# assumes %bx has loop count in it

.main
.top
# critical section
mov 2000, %ax  # get 'value' at address 2000
add $1, %ax    # increment it
mov %ax, 2000  # store it back

# see if we're still looping
sub  $1, %bx
test $0, %bx
jgt .top

halt
```

可以看到该程序与之前不同的地方在于 增加了访问临界区

将临界区的内容取出 加一 并存入临界区

由于只有一个线程 所以运行结果应为 地址内的数据由0变为1

```
-> % ./x86.py -p looping-race-nolock.s -t 1 -M 2000 -c
ARG seed 0
ARG numthreads 1
ARG program looping-race-nolock.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace
ARG cctrace False
ARG printstats False
ARG verbose False

 2000          Thread 0
    0
    0   1000 mov 2000, %ax
    0   1001 add $1, %ax
    1   1002 mov %ax, 2000
    1   1003 sub  $1, %bx
    1   1004 test $0, %bx
    1   1005 jgt .top
    1   1006 halt
```

28.1

首先分析flag.s

```
.var flag
.var count

.main
.top

.acquire
mov  flag, %ax      # get flag
test $0, %ax        # if we get 0 back: lock is free!
jne  .acquire       # if not, try again
mov  $1, flag       # store 1 into flag

# critical section
mov  count, %ax     # get the value at the address
add  $1, %ax        # increment it
mov  %ax, count     # store it back

# release lock
mov  $0, flag       # clear the flag now

# see if we're still looping
sub  $1, %bx
test $0, %bx
jgt .top

halt
```

可以看到 该汇编程序通过检查一个flag变量来判断锁的状态

拥有锁的线程将flag设为1 其他线程需要等待该线程释放锁后 flag为0 才能获得锁

运行结果如下：

```
-> % ./x86.py -p flag.s
ARG seed 0
ARG numthreads 2
ARG program flag.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG procsched
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace
ARG regtrace
ARG cctrace False
ARG printstats False
ARG verbose False


       Thread 0                Thread 1

1000 mov  flag, %ax
1001 test $0, %ax
1002 jne  .acquire
1003 mov  $1, flag
1004 mov  count, %ax
1005 add  $1, %ax
1006 mov  %ax, count
1007 mov  $0, flag
1008 sub  $1, %bx
1009 test $0, %bx
1010 jgt .top
1011 halt
----- Halt;Switch -----  ----- Halt;Switch -----
                         1000 mov  flag, %ax
                         1001 test $0, %ax
                         1002 jne  .acquire
                         1003 mov  $1, flag
                         1004 mov  count, %ax
                         1005 add  $1, %ax
                         1006 mov  %ax, count
                         1007 mov  $0, flag
                         1008 sub  $1, %bx
                         1009 test $0, %bx
                         1010 jgt .top
                         1011 halt
```

28.2

由于默认的中断频率是50条指令

所以运行顺序是先运行线程0后运行线程1

首先线程0获得锁 flag变成1

线程0将count值加1

线程0释放锁 flag变成0

然后线程1获得锁 flag变成1

线程1将count值加1

线程1释放锁 flag变成0

实际运行结果如下：

```
-> % ./x86.py -p flag.s -R ax,bx -M flag,count -c
ARG seed 0
ARG numthreads 2
ARG program flag.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG procsched
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace flag,count
ARG regtrace ax,bx
ARG cctrace False
ARG printstats False
ARG verbose False


 flag count      ax    bx          Thread 0                Thread 1

    0     0       0     0
    0     0       0     0   1000 mov  flag, %ax
    0     0       0     0   1001 test $0, %ax
    0     0       0     0   1002 jne  .acquire
    1     0       0     0   1003 mov  $1, flag
    1     0       0     0   1004 mov  count, %ax
    1     0       1     0   1005 add  $1, %ax
    1     1       1     0   1006 mov  %ax, count
    0     1       1     0   1007 mov  $0, flag
    0     1       1    -1   1008 sub  $1, %bx
    0     1       1    -1   1009 test $0, %bx
    0     1       1    -1   1010 jgt .top
    0     1       1    -1   1011 halt
    0     1       0     0   ----- Halt;Switch -----  ----- Halt;Switch -----
    0     1       0     0                            1000 mov  flag, %ax
    0     1       0     0                            1001 test $0, %ax
    0     1       0     0                            1002 jne  .acquire
    1     1       0     0                            1003 mov  $1, flag
    1     1       1     0                            1004 mov  count, %ax
    1     1       2     0                            1005 add  $1, %ax
    1     2       2     0                            1006 mov  %ax, count
    0     2       2     0                            1007 mov  $0, flag
    0     2       2    -1                            1008 sub  $1, %bx
    0     2       2    -1                            1009 test $0, %bx
    0     2       2    -1                            1010 jgt .top
    0     2       2    -1                            1011 halt
```

28.3

由于默认的中断频率是50条指令

所以运行顺序是先运行线程0后运行线程1

首先线程0获得锁 flag变成1

线程0将count值加1

线程0释放锁 flag变成0

重复一次上面的动作

然后线程1获得锁 flag变成1

线程1将count值加1

线程1释放锁 flag变成0

重复一次上面的动作

所以经过两个线程先后的操作

count的值变为4

```
-> % ./x86.py -p flag.s -R ax,bx -M flag,count -a bx=2,bx=2 -c
ARG seed 0
ARG numthreads 2
ARG program flag.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG procsched
ARG argv bx=2,bx=2
ARG load address 1000
ARG memsize 128
ARG memtrace flag,count
ARG regtrace ax,bx
ARG cctrace False
ARG printstats False
ARG verbose False


 flag count      ax    bx          Thread 0                Thread 1

    0     0       0     2
    0     0       0     2   1000 mov  flag, %ax
    0     0       0     2   1001 test $0, %ax
    0     0       0     2   1002 jne  .acquire
    1     0       0     2   1003 mov  $1, flag
    1     0       0     2   1004 mov  count, %ax
    1     0       1     2   1005 add  $1, %ax
    1     1       1     2   1006 mov  %ax, count
    0     1       1     2   1007 mov  $0, flag
    0     1       1     1   1008 sub  $1, %bx
    0     1       1     1   1009 test $0, %bx
    0     1       1     1   1010 jgt .top
    0     1       0     1   1000 mov  flag, %ax
    0     1       0     1   1001 test $0, %ax
    0     1       0     1   1002 jne  .acquire
```

```
     1     1       0     1   1003 mov  $1, flag
     1     1       1     1   1004 mov  count, %ax
     1     1       2     1   1005 add  $1, %ax
     1     2       2     1   1006 mov  %ax, count
     0     2       2     1   1007 mov  $0, flag
     0     2       2     0   1008 sub  $1, %bx
     0     2       2     0   1009 test $0, %bx
     0     2       2     0   1010 jgt .top
     0     2       2     0   1011 halt
     0     2       0     2   ----- Halt;Switch -----  ----- Halt;Switch -----
     0     2       0     2                            1000 mov  flag, %ax
     0     2       0     2                            1001 test $0, %ax
     0     2       0     2                            1002 jne  .acquire
     1     2       0     2                            1003 mov  $1, flag
     1     2       2     2                            1004 mov  count, %ax
     1     2       3     2                            1005 add  $1, %ax
     1     3       3     2                            1006 mov  %ax, count
     0     3       3     2                            1007 mov  $0, flag
     0     3       3     1                            1008 sub  $1, %bx
     0     3       3     1                            1009 test $0, %bx
     0     3       3     1                            1010 jgt .top
     0     3       0     1                            1000 mov  flag, %ax
     0     3       0     1                            1001 test $0, %ax
     0     3       0     1                            1002 jne  .acquire
     1     3       0     1                            1003 mov  $1, flag
     1     3       3     1                            1004 mov  count, %ax
     1     3       4     1                            1005 add  $1, %ax
     1     4       4     1                            1006 mov  %ax, count
     0     4       4     1                            1007 mov  $0, flag
     0     4       4     0                            1008 sub  $1, %bx
     0     4       4     0                            1009 test $0, %bx
     0     4       4     0                            1010 jgt .top
     0     4       4     0                            1011 halt
```

28.4

```
.var flag
.var count

.main
.top

.acquire
mov  flag, %ax      # get flag
test $0, %ax        # if we get 0 back: lock is free!
jne  .acquire       # if not, try again
mov  $1, flag       # store 1 into flag

# critical section
mov  count, %ax     # get the value at the address
add  $1, %ax        # increment it
mov  %ax, count     # store it back

# release lock
```

```
mov  $0, flag       # clear the flag now

# see if we're still looping
sub  $1, %bx
test $0, %bx
jgt .top

halt
```

仔细分析flag.s 可以发现

如果在 jne .acquire 进行中断

即破坏了获取锁这个操作的原子性

可以让两个线程同时获得锁

所以构造 中断频率为3

运行结果如下：

```
-> % ./x86.py -p flag.s -R ax,bx -M flag,count -a bx=2,bx=2 -c -i 3
ARG seed 0
ARG numthreads 2
ARG program flag.s
ARG interrupt frequency 3
ARG interrupt randomness False
ARG procsched
ARG argv bx=2,bx=2
ARG load address 1000
ARG memsize 128
ARG memtrace flag,count
ARG regtrace ax,bx
ARG cctrace False
ARG printstats False
ARG verbose False


 flag count      ax    bx          Thread 0                Thread 1

    0     0       0     2
    0     0       0     2   1000 mov  flag, %ax
    0     0       0     2   1001 test $0, %ax
    0     0       0     2   1002 jne  .acquire
    0     0       0     2   ------ Interrupt ------  ------ Interrupt ------
    0     0       0     2                            1000 mov  flag, %ax
    0     0       0     2                            1001 test $0, %ax
    0     0       0     2                            1002 jne  .acquire
    0     0       0     2   ------ Interrupt ------  ------ Interrupt ------
    1     0       0     2   1003 mov  $1, flag
    1     0       0     2   1004 mov  count, %ax
    1     0       1     2   1005 add  $1, %ax
    1     0       0     2   ------ Interrupt ------  ------ Interrupt ------
    1     0       0     2                            1003 mov  $1, flag
    1     0       0     2                            1004 mov  count, %ax
    1     0       1     2                            1005 add  $1, %ax
    1     0       1     2   ------ Interrupt ------  ------ Interrupt ------
    1     1       1     2   1006 mov  %ax, count
    0     1       1     2   1007 mov  $0, flag
```

```
     0     1       1     1    1008 sub  $1, %bx
     0     1       1     2    ------ Interrupt ------  ------ Interrupt ------
     0     1       1     2                             1006 mov  %ax, count
     0     1       1     2                             1007 mov  $0, flag
     0     1       1     1                             1008 sub  $1, %bx
     0     1       1     1    ------ Interrupt ------  ------ Interrupt ------
     0     1       1     1    1009 test $0, %bx
     0     1       1     1    1010 jgt .top
     0     1       0     1    1000 mov  flag, %ax
     0     1       1     1    ------ Interrupt ------  ------ Interrupt ------
     0     1       1     1                             1009 test $0, %bx
     0     1       1     1                             1010 jgt .top
     0     1       0     1                             1000 mov  flag, %ax
     0     1       0     1    ------ Interrupt ------  ------ Interrupt ------
     0     1       0     1    1001 test $0, %ax
     0     1       0     1    1002 jne  .acquire
     1     1       0     1    1003 mov  $1, flag
     1     1       0     1    ------ Interrupt ------  ------ Interrupt ------
     1     1       0     1                             1001 test $0, %ax
     1     1       0     1                             1002 jne  .acquire
     1     1       0     1                             1003 mov  $1, flag
     1     1       0     1    ------ Interrupt ------  ------ Interrupt ------
     1     1       1     1    1004 mov  count, %ax
     1     1       2     1    1005 add  $1, %ax
     1     2       2     1    1006 mov  %ax, count
     1     2       0     1    ------ Interrupt ------  ------ Interrupt ------
     1     2       2     1                             1004 mov  count, %ax
     1     2       3     1                             1005 add  $1, %ax
     1     3       3     1                             1006 mov  %ax, count
     1     3       2     1    ------ Interrupt ------  ------ Interrupt ------
     0     3       2     1    1007 mov  $0, flag
     0     3       2     0    1008 sub  $1, %bx
     0     3       2     0    1009 test $0, %bx
     0     3       3     1    ------ Interrupt ------  ------ Interrupt ------
     0     3       3     1                             1007 mov  $0, flag
     0     3       3     0                             1008 sub  $1, %bx
     0     3       3     0                             1009 test $0, %bx
     0     3       2     0    ------ Interrupt ------  ------ Interrupt ------
     0     3       2     0    1010 jgt .top
     0     3       2     0    1011 halt
     0     3       3     0    ----- Halt;Switch -----  ----- Halt;Switch -----
     0     3       3     0                             1010 jgt .top
     0     3       3     0    ------ Interrupt ------  ------ Interrupt ------
     0     3       3     0                             1011 halt
```

可以看到 程序执行结果出现了错误 最后count的值变成了3

说明这种实现锁的方式存在问题

如果恰好在检查锁变量值之后发生中断 会导致多个线程同时获得锁

28.5

首先查看test-and-set.s

```
.var mutex
```

```
.var count

.main
.top

.acquire
mov  $1, %ax
xchg %ax, mutex     # atomic swap of 1 and mutex
test $0, %ax        # if we get 0 back: lock is free!
jne  .acquire       # if not, try again

# critical section
mov  count, %ax     # get the value at the address
add  $1, %ax        # increment it
mov  %ax, count     # store it back

# release lock
mov  $0, mutex

# see if we're still looping
sub  $1, %bx
test $0, %bx
jgt .top

halt
```

可以看到这个汇编程序和之前不一样的地方在于 使用了原子性的操作xchg

如果ax的值为0 说明mutex的值为0 说明锁可用

硬件层面保证了xchg操作的原子性 不会被线程切换而中断

查找xchg的资料如下

XCHG指令，双操作数指令，用于交换src和dest操作数的内容。其中，src和dest可以是两个通用寄存器，也可以是一个寄存器和一个memory位置。在XCHG执行期间，memory操作数被引用时，处理器自动实现locking protocol，不依赖LOCK prefix或IOPL字段（I/O privilege level field，EFR寄存器中的IOPL字段）的值。（参考locking protocol机制中的LOCK prefix描述）。

运行结果如下：

```
-> % ./x86.py -p test-and-set.s -R ax,bx -M count -a bx=2,bx=2 -c -i 3
ARG seed 0
ARG numthreads 2
ARG program test-and-set.s
ARG interrupt frequency 3
ARG interrupt randomness False
ARG procsched
ARG argv bx=2,bx=2
ARG load address 1000
ARG memsize 128
ARG memtrace count
ARG regtrace ax,bx
ARG cctrace False
ARG printstats False
ARG verbose False
```

```
count      ax    bx          Thread 0                Thread 1

    0       0     2
    0       1     2   1000 mov  $1, %ax
    0       0     2   1001 xchg %ax, mutex
    0       0     2   1002 test $0, %ax
    0       0     2   ------ Interrupt ------  ------ Interrupt ------
    0       1     2                            1000 mov  $1, %ax
    0       1     2                            1001 xchg %ax, mutex
    0       1     2                            1002 test $0, %ax
    0       0     2   ------ Interrupt ------  ------ Interrupt ------
    0       0     2   1003 jne  .acquire
    0       0     2   1004 mov  count, %ax
    0       1     2   1005 add  $1, %ax
    0       1     2   ------ Interrupt ------  ------ Interrupt ------
    0       1     2                            1003 jne  .acquire
    0       1     2                            1000 mov  $1, %ax
    0       1     2                            1001 xchg %ax, mutex
    0       1     2   ------ Interrupt ------  ------ Interrupt ------
    1       1     2   1006 mov  %ax, count
    1       1     2   1007 mov  $0, mutex
    1       1     1   1008 sub  $1, %bx
    1       1     2   ------ Interrupt ------  ------ Interrupt ------
    1       1     2                            1002 test $0, %ax
    1       1     2                            1003 jne  .acquire
    1       1     2                            1000 mov  $1, %ax
    1       1     1   ------ Interrupt ------  ------ Interrupt ------
    1       1     1   1009 test $0, %bx
    1       1     1   1010 jgt .top
    1       1     1   1000 mov  $1, %ax
    1       1     2   ------ Interrupt ------  ------ Interrupt ------
    1       0     2                            1001 xchg %ax, mutex
    1       0     2                            1002 test $0, %ax
    1       0     2                            1003 jne  .acquire
    1       1     1   ------ Interrupt ------  ------ Interrupt ------
    1       1     1   1001 xchg %ax, mutex
    1       1     1   1002 test $0, %ax
    1       1     1   1003 jne  .acquire
    1       0     2   ------ Interrupt ------  ------ Interrupt ------
    1       1     2                            1004 mov  count, %ax
    1       2     2                            1005 add  $1, %ax
    2       2     2                            1006 mov  %ax, count
    2       1     1   ------ Interrupt ------  ------ Interrupt ------
    2       1     1   1000 mov  $1, %ax
    2       1     1   1001 xchg %ax, mutex
    2       1     1   1002 test $0, %ax
    2       2     2   ------ Interrupt ------  ------ Interrupt ------
    2       2     2                            1007 mov  $0, mutex
    2       2     1                            1008 sub  $1, %bx
    2       2     1                            1009 test $0, %bx
    2       1     1   ------ Interrupt ------  ------ Interrupt ------
    2       1     1   1003 jne  .acquire
    2       1     1   1000 mov  $1, %ax
    2       0     1   1001 xchg %ax, mutex
    2       2     1   ------ Interrupt ------  ------ Interrupt ------
    2       2     1                            1010 jgt .top
    2       1     1                            1000 mov  $1, %ax
    2       1     1                         1001 xchg %ax, mutex
    2       0     1   ------ Interrupt ------  ------ Interrupt ------
    2       0     1   1002 test $0, %ax
    2       0     1   1003 jne  .acquire
    2       2     1   1004 mov  count, %ax
    2       1     1   ------ Interrupt ------  ------ Interrupt ------
    2       1     1                         1002 test $0, %ax
    2       1     1                         1003 jne  .acquire
    2       1     1                         1000 mov  $1, %ax
    2       2     1   ------ Interrupt ------  ------ Interrupt ------
    2       3     1   1005 add  $1, %ax
    3       3     1   1006 mov  %ax, count
    3       3     1   1007 mov  $0, mutex
    3       1     1   ------ Interrupt ------  ------ Interrupt ------
    3       0     1                         1001 xchg %ax, mutex
    3       0     1                         1002 test $0, %ax
    3       0     1                         1003 jne  .acquire
    3       3     1   ------ Interrupt ------  ------ Interrupt ------
    3       3     0   1008 sub  $1, %bx
    3       3     0   1009 test $0, %bx
    3       3     0   1010 jgt .top
    3       0     1   ------ Interrupt ------  ------ Interrupt ------
    3       3     1                         1004 mov  count, %ax
    3       4     1                         1005 add  $1, %ax
    4       4     1                         1006 mov  %ax, count
    4       3     0   ------ Interrupt ------  ------ Interrupt ------
    4       3     0   1011 halt
    4       4     1   ----- Halt;Switch -----  ----- Halt;Switch -----
    4       4     1                         1007 mov  $0, mutex
    4       4     0                         1008 sub  $1, %bx
    4       4     0   ------ Interrupt ------  ------ Interrupt ------
    4       4     0                         1009 test $0, %bx
    4       4     0                         1010 jgt .top
    4       4     0                         1011 halt
```

可以看到 在使用xchg操作之后 即使中断频率为3 程序也能正常运行

因为xchg指令不可分 所以不会出现之前flag变量出现的问题

28.6

经过多次调整中断频率发现

程序都可以按照预期运行

CPU使用频率高的情况：在获取锁的几条指令中不断循环会导致cpu空转 浪费CPU资源

下面是中断频率为10的运行结果：

```
-> % ./x86.py -p test-and-set.s -i 10 -R ax,bx -M mutex,count -a bx=5 -c

ARG seed 0
ARG numthreads 2
ARG program test-and-set.s
ARG interrupt frequency 10
ARG interrupt randomness False
ARG procsched
ARG argv bx=5
ARG load address 1000
ARG memsize 128
ARG memtrace mutex,count
ARG regtrace ax,bx
ARG cctrace False
ARG printstats False
ARG verbose False


mutex count      ax     bx          Thread 0                Thread 1

    0     0       0     5
    0     0       1     5   1000 mov  $1, %ax
    1     0       0     5   1001 xchg %ax, mutex
    1     0       0     5   1002 test $0, %ax
    1     0       0     5   1003 jne  .acquire
    1     0       0     5   1004 mov  count, %ax
    1     0       1     5   1005 add  $1, %ax
    1     1       1     5   1006 mov  %ax, count
    0     1       1     5   1007 mov  $0, mutex
    0     1       1     4   1008 sub  $1, %bx
    0     1       1     4   1009 test $0, %bx
    0     1       0     5   ------ Interrupt ------  ------ Interrupt ------
    0     1       1     5                            1000 mov  $1, %ax
    1     1       0     5                            1001 xchg %ax, mutex
    1     1       0     5                            1002 test $0, %ax
    1     1       0     5                            1003 jne  .acquire
    1     1       1     5                            1004 mov  count, %ax
    1     1       2     5                            1005 add  $1, %ax
    1     2       2     5                            1006 mov  %ax, count
    0     2       2     5                            1007 mov  $0, mutex
    0     2       2     4                            1008 sub  $1, %bx
    0     2       2     4                            1009 test $0, %bx
    0     2       1     4   ------ Interrupt ------  ------ Interrupt ------
    0     2       1     4   1010 jgt .top
    0     2       1     4   1000 mov  $1, %ax
    1     2       0     4   1001 xchg %ax, mutex
    1     2       0     4   1002 test $0, %ax
    1     2       0     4   1003 jne  .acquire
    1     2       2     4   1004 mov  count, %ax
    1     2       3     4   1005 add  $1, %ax
    1     3       3     4   1006 mov  %ax, count
    0     3       3     4   1007 mov  $0, mutex
    0     3       3     3   1008 sub  $1, %bx
    0     3       2     4   ------ Interrupt ------  ------ Interrupt ------
    0     3       2     4                            1010 jgt .top
    0     3       1     4                            1000 mov  $1, %ax
    1     3       0     4                            1001 xchg %ax, mutex
    1     3       0     4                            1002 test $0, %ax
    1     3       0     4                            1003 jne  .acquire
    1     3       3     4                            1004 mov  count, %ax
    1     3       4     4                            1005 add  $1, %ax
    1     4       4     4                            1006 mov  %ax, count
    0     4       4     4                            1007 mov  $0, mutex
    0     4       4     3                            1008 sub  $1, %bx
    0     4       3     3   ------ Interrupt ------  ------ Interrupt ------
    0     4       3     3   1009 test $0, %bx
    0     4       3     3   1010 jgt .top
```

```
     0      4        1      3   1000 mov  $1, %ax
     1      4        0      3   1001 xchg %ax, mutex
     1      4        0      3   1002 test $0, %ax
     1      4        0      3   1003 jne  .acquire
     1      4        4      3   1004 mov  count, %ax
     1      4        5      3   1005 add  $1, %ax
     1      5        5      3   1006 mov  %ax, count
     0      5        5      3   1007 mov  $0, mutex
     0      5        4      3   ------ Interrupt ------  ------ Interrupt ------
     0      5        4      3                           1009 test $0, %bx
     0      5        4      3                           1010 jgt .top
     0      5        1      3                           1000 mov  $1, %ax
     1      5        0      3                           1001 xchg %ax, mutex
     1      5        0      3                           1002 test $0, %ax
     1      5        0      3                           1003 jne  .acquire
     1      5        5      3                           1004 mov  count, %ax
     1      5        6      3                           1005 add  $1, %ax
     1      6        6      3                           1006 mov  %ax, count
     0      6        6      3                           1007 mov  $0, mutex
     0      6        5      3   ------ Interrupt ------  ------ Interrupt ------
     0      6        5      2   1008 sub  $1, %bx
     0      6        5      2   1009 test $0, %bx
     0      6        5      2   1010 jgt .top
     0      6        1      2   1000 mov  $1, %ax
     1      6        0      2   1001 xchg %ax, mutex
     1      6        0      2   1002 test $0, %ax
     1      6        0      2   1003 jne  .acquire
     1      6        6      2   1004 mov  count, %ax
     1      6        7      2   1005 add  $1, %ax
     1      7        7      2   1006 mov  %ax, count
     1      7        6      3   ------ Interrupt ------  ------ Interrupt ------
     1      7        6      2                           1008 sub  $1, %bx
     1      7        6      2                           1009 test $0, %bx
     1      7        6      2                           1010 jgt .top
     1      7        1      2                           1000 mov  $1, %ax
     1      7        1      2                           1001 xchg %ax, mutex
     1      7        1      2                           1002 test $0, %ax
     1      7        1      2                           1003 jne  .acquire
     1      7        1      2                           1000 mov  $1, %ax
     1      7        1      2                           1001 xchg %ax, mutex
     1      7        1      2                           1002 test $0, %ax
     1      7        7      2   ------ Interrupt ------  ------ Interrupt ------
     0      7        7      2   1007 mov  $0, mutex
     0      7        7      1   1008 sub  $1, %bx
     0      7        7      1   1009 test $0, %bx
     0      7        7      1   1010 jgt .top
     0      7        1      1   1000 mov  $1, %ax
     1      7        0      1   1001 xchg %ax, mutex
     1      7        0      1   1002 test $0, %ax
     1      7        0      1   1003 jne  .acquire
     1      7        7      1   1004 mov  count, %ax
     1      7        8      1   1005 add  $1, %ax
     1      7        1      2   ------ Interrupt ------  ------ Interrupt ------
     1      7        1      2                           1003 jne  .acquire
     1      7        1      2                           1000 mov  $1, %ax
     1      7        1      2                           1001 xchg %ax, mutex
     1      7        1      2                           1002 test $0, %ax
     1      7        1      2                           1003 jne  .acquire
```

```
      1     7       1     2                          1000 mov  $1, %ax
      1     7       1     2                          1001 xchg %ax, mutex
      1     7       1     2                          1002 test $0, %ax
      1     7       1     2                          1003 jne  .acquire
      1     7       1     2                          1000 mov  $1, %ax
      1     7       8     1  ------ Interrupt ------  ------ Interrupt ------
      1     8       8     1  1006 mov  %ax, count
      0     8       8     1  1007 mov  $0, mutex
      0     8       8     0  1008 sub  $1, %bx
      0     8       8     0  1009 test $0, %bx
      0     8       8     0  1010 jgt .top
      0     8       8     0  1011 halt
      0     8       1     2  ----- Halt;Switch -----  ----- Halt;Switch -----
      1     8       0     2                          1001 xchg %ax, mutex
      1     8       0     2                          1002 test $0, %ax
      1     8       0     2                          1003 jne  .acquire
      1     8       8     2                          1004 mov  count, %ax
      1     8       8     2  ------ Interrupt ------  ------ Interrupt ------
      1     8       9     2                          1005 add  $1, %ax
      1     9       9     2                          1006 mov  %ax, count
      0     9       9     2                          1007 mov  $0, mutex
      0     9       9     1                          1008 sub  $1, %bx
      0     9       9     1                          1009 test $0, %bx
      0     9       9     1                          1010 jgt .top
      0     9       1     1                          1000 mov  $1, %ax
      1     9       0     1                          1001 xchg %ax, mutex
      1     9       0     1                          1002 test $0, %ax
      1     9       0     1                          1003 jne  .acquire
      1     9       0     1  ------ Interrupt ------  ------ Interrupt ------
      1     9       9     1                          1004 mov  count, %ax
      1     9      10     1                          1005 add  $1, %ax
      1    10      10     1                          1006 mov  %ax, count
      0    10      10     1                          1007 mov  $0, mutex
      0    10      10     0                          1008 sub  $1, %bx
      0    10      10     0                          1009 test $0, %bx
      0    10      10     0                          1010 jgt .top
      0    10      10     0                          1011 halt
```

28.7

构造一个获取锁的序列111000111000

运行结果如下：

```
-> % ./x86.py -p test-and-set.s -i 10 -R ax,bx -M mutex,count -a bx=5 -P
111000111000 -c

ARG seed 0
ARG numthreads 2
ARG program test-and-set.s
ARG interrupt frequency 10
ARG interrupt randomness False
ARG procsched 111000111000
ARG argv bx=5
ARG load address 1000
ARG memsize 128
ARG memtrace mutex,count
ARG regtrace ax,bx
ARG cctrace False
ARG printstats False
ARG verbose False


mutex count      ax    bx          Thread 0                Thread 1

    0     0       0     5
    0     0       1     5                              1000 mov  $1, %ax
    1     0       0     5                              1001 xchg %ax, mutex
    1     0       0     5                              1002 test $0, %ax
    1     0       0     5   ------ Interrupt ------    ------ Interrupt ------
    1     0       1     5   1000 mov  $1, %ax
    1     0       1     5   1001 xchg %ax, mutex
    1     0       1     5   1002 test $0, %ax
    1     0       0     5   ------ Interrupt ------    ------ Interrupt ------
    1     0       0     5                              1003 jne  .acquire
    1     0       0     5                              1004 mov  count, %ax
    1     0       1     5                              1005 add  $1, %ax
    1     0       1     5   ------ Interrupt ------    ------ Interrupt ------
    1     0       1     5   1003 jne  .acquire
    1     0       1     5   1000 mov  $1, %ax
    1     0       1     5   1001 xchg %ax, mutex
    1     0       1     5   ------ Interrupt ------    ------ Interrupt ------
    1     1       1     5                              1006 mov  %ax, count
    0     1       1     5                              1007 mov  $0, mutex
    0     1       1     4                              1008 sub  $1, %bx
    0     1       1     5   ------ Interrupt ------    ------ Interrupt ------
    0     1       1     5   1002 test $0, %ax
    0     1       1     5   1003 jne  .acquire
    0     1       1     5   1000 mov  $1, %ax
    0     1       1     4   ------ Interrupt ------    ------ Interrupt ------
    0     1       1     4                              1009 test $0, %bx
    0     1       1     4                              1010 jgt .top
    0     1       1     4                              1000 mov  $1, %ax
    0     1       1     5   ------ Interrupt ------    ------ Interrupt ------
    1     1       0     5   1001 xchg %ax, mutex
    1     1       0     5   1002 test $0, %ax
    1     1       0     5   1003 jne  .acquire
    1     1       1     4   ------ Interrupt ------    ------ Interrupt ------
    1     1       1     4                              1001 xchg %ax, mutex
    1     1       1     4                              1002 test $0, %ax
    1     1       1     4                              1003 jne  .acquire
    1     1       0     5   ------ Interrupt ------    ------ Interrupt ------
    1     1       1     5   1004 mov  count, %ax
    1     1       2     5   1005 add  $1, %ax
    1     2       2     5   1006 mov  %ax, count
    1     2       1     4   ------ Interrupt ------    ------ Interrupt ------
    1     2       1     4                              1000 mov  $1, %ax
    1     2       1     4                              1001 xchg %ax, mutex
    1     2       1     4                              1002 test $0, %ax
    1     2       2     5   ------ Interrupt ------    ------ Interrupt ------
    0     2       2     5   1007 mov  $0, mutex
    0     2       2     4   1008 sub  $1, %bx
    0     2       2     4   1009 test $0, %bx
    0     2       1     4   ------ Interrupt ------    ------ Interrupt ------
    0     2       1     4                              1003 jne  .acquire
    0     2       1     4                              1000 mov  $1, %ax
    1     2       0     4                              1001 xchg %ax, mutex
    1     2       2     4   ------ Interrupt ------   ------ Interrupt ------
    1     2       2     4   1010 jgt .top
    1     2       1     4   1000 mov  $1, %ax
    1     2       1     4   1001 xchg %ax, mutex
    1     2       0     4   ------ Interrupt ------   ------ Interrupt ------
    1     2       0     4                              1002 test $0, %ax
    1     2       0     4                              1003 jne  .acquire
    1     2       2     4                              1004 mov  count, %ax
    1     2       1     4   ------ Interrupt ------   ------ Interrupt ------
    1     2       1     4   1002 test $0, %ax
    1     2       1     4   1003 jne  .acquire
    1     2       1     4   1000 mov  $1, %ax
    1     2       2     4   ------ Interrupt ------   ------ Interrupt ------
    1     2       3     4                              1005 add  $1, %ax
    1     3       3     4                              1006 mov  %ax, count
    0     3       3     4                              1007 mov  $0, mutex
    0     3       1     4   ------ Interrupt ------   ------ Interrupt ------
    1     3       0     4   1001 xchg %ax, mutex
    1     3       0     4   1002 test $0, %ax
    1     3       0     4   1003 jne  .acquire
    1     3       3     4   ------ Interrupt ------   ------ Interrupt ------
    1     3       3     3                              1008 sub  $1, %bx
    1     3       3     3                              1009 test $0, %bx
    1     3       3     3                              1010 jgt .top
    1     3       0     4   ------ Interrupt ------   ------ Interrupt ------
    1     3       3     4   1004 mov  count, %ax
    1     3       4     4   1005 add  $1, %ax
    1     4       4     4   1006 mov  %ax, count
    1     4       3     3   ------ Interrupt ------   ------ Interrupt ------
    1     4       1     3                              1000 mov  $1, %ax
    1     4       1     3                              1001 xchg %ax, mutex
    1     4       1     3                              1002 test $0, %ax
    1     4       4     4   ------ Interrupt ------   ------ Interrupt ------
    0     4       4     4   1007 mov  $0, mutex
    0     4       4     3   1008 sub  $1, %bx
    0     4       4     3   1009 test $0, %bx
    0     4       1     3   ------ Interrupt ------   ------ Interrupt ------
    0     4       1     3                              1003 jne  .acquire
    0     4       1     3                              1000 mov  $1, %ax
    1     4       0     3                              1001 xchg %ax, mutex
    1     4       4     3   ------ Interrupt ------   ------ Interrupt ------
    1     4       4     3   1010 jgt .top
    1     4       1     3   1000 mov  $1, %ax
    1     4       1     3   1001 xchg %ax, mutex
    1     4       0     3   ------ Interrupt ------   ------ Interrupt ------
    1     4       0     3                              1002 test $0, %ax
    1     4       0     3                              1003 jne  .acquire
    1     4       4     3                              1004 mov  count, %ax
    1     4       1     3   ------ Interrupt ------   ------ Interrupt ------
    1     4       1     3   1002 test $0, %ax
    1     4       1     3   1003 jne  .acquire
    1     4       1     3   1000 mov  $1, %ax
    1     4       4     3   ------ Interrupt ------   ------ Interrupt ------
    1     4       5     3                              1005 add  $1, %ax
    1     5       5     3                              1006 mov  %ax, count
    0     5       5     3                              1007 mov  $0, mutex
    0     5       1     3   ------ Interrupt ------  ------ Interrupt ------
    1     5       0     3   1001 xchg %ax, mutex
    1     5       0     3   1002 test $0, %ax
    1     5       0     3   1003 jne  .acquire
    1     5       5     3   ------ Interrupt ------  ------ Interrupt ------
    1     5       5     2                            1008 sub  $1, %bx
    1     5       5     2                            1009 test $0, %bx
    1     5       5     2                            1010 jgt .top
    1     5       0     3   ------ Interrupt ------  ------ Interrupt ------
    1     5       5     3   1004 mov  count, %ax
    1     5       6     3   1005 add  $1, %ax
    1     6       6     3   1006 mov  %ax, count
    1     6       5     2   ------ Interrupt ------  ------ Interrupt ------
    1     6       1     2                            1000 mov  $1, %ax
    1     6       1     2                            1001 xchg %ax, mutex
    1     6       1     2                            1002 test $0, %ax
    1     6       6     3   ------ Interrupt ------  ------ Interrupt ------
    0     6       6     3   1007 mov  $0, mutex
    0     6       6     2   1008 sub  $1, %bx
    0     6       6     2   1009 test $0, %bx
    0     6       1     2   ------ Interrupt ------  ------ Interrupt ------
    0     6       1     2                            1003 jne  .acquire
    0     6       1     2                            1000 mov  $1, %ax
    1     6       0     2                            1001 xchg %ax, mutex
    1     6       6     2   ------ Interrupt ------  ------ Interrupt ------
    1     6       6     2   1010 jgt .top
    1     6       1     2   1000 mov  $1, %ax
    1     6       1     2   1001 xchg %ax, mutex
    1     6       0     2   ------ Interrupt ------  ------ Interrupt ------
    1     6       0     2                            1002 test $0, %ax
    1     6       0     2                            1003 jne  .acquire
    1     6       6     2                            1004 mov  count, %ax
    1     6       1     2   ------ Interrupt ------  ------ Interrupt ------
    1     6       1     2   1002 test $0, %ax
    1     6       1     2   1003 jne  .acquire
    1     6       1     2   1000 mov  $1, %ax
    1     6       6     2   ------ Interrupt ------  ------ Interrupt ------
    1     6       7     2                            1005 add  $1, %ax
    1     7       7     2                            1006 mov  %ax, count
    0     7       7     2                            1007 mov  $0, mutex
    0     7       1     2   ------ Interrupt ------  ------ Interrupt ------
    1     7       0     2   1001 xchg %ax, mutex
    1     7       0     2   1002 test $0, %ax
    1     7       0     2   1003 jne  .acquire
    1     7       7     2   ------ Interrupt ------  ------ Interrupt ------
    1     7       7     1                            1008 sub  $1, %bx
    1     7       7     1                            1009 test $0, %bx
    1     7       7     1                            1010 jgt .top
    1     7       0     2   ------ Interrupt ------  ------ Interrupt ------
    1     7       7     2   1004 mov  count, %ax
    1     7       8     2   1005 add  $1, %ax
    1     8       8     2   1006 mov  %ax, count
    1     8       7     1   ------ Interrupt ------  ------ Interrupt ------
    1     8       1     1                            1000 mov  $1, %ax
    1     8       1     1                            1001 xchg %ax, mutex
    1     8       1     1                            1002 test $0, %ax
    1     8       8     2   ------ Interrupt ------  ------ Interrupt ------
    0     8       8     2   1007 mov  $0, mutex
```

```
     0     8       8     1    1008 sub  $1, %bx
     0     8       8     1    1009 test $0, %bx
     0     8       1     1    ------ Interrupt ------  ------ Interrupt ------
     0     8       1     1                             1003 jne  .acquire
     0     8       1     1                             1000 mov  $1, %ax
     1     8       0     1                             1001 xchg %ax, mutex
     1     8       8     1    ------ Interrupt ------  ------ Interrupt ------
     1     8       8     1    1010 jgt .top
     1     8       1     1    1000 mov  $1, %ax
     1     8       1     1    1001 xchg %ax, mutex
     1     8       0     1    ------ Interrupt ------  ------ Interrupt ------
     1     8       0     1                             1002 test $0, %ax
     1     8       0     1                             1003 jne  .acquire
     1     8       8     1                             1004 mov  count, %ax
     1     8       1     1    ------ Interrupt ------  ------ Interrupt ------
     1     8       1     1    1002 test $0, %ax
     1     8       1     1    1003 jne  .acquire
     1     8       1     1    1000 mov  $1, %ax
     1     8       8     1    ------ Interrupt ------  ------ Interrupt ------
     1     8       9     1                             1005 add  $1, %ax
     1     9       9     1                             1006 mov  %ax, count
     0     9       9     1                             1007 mov  $0, mutex
     0     9       1     1    ------ Interrupt ------  ------ Interrupt ------
     1     9       0     1    1001 xchg %ax, mutex
     1     9       0     1    1002 test $0, %ax
     1     9       0     1    1003 jne  .acquire
     1     9       9     1    ------ Interrupt ------  ------ Interrupt ------
     1     9       9     0                             1008 sub  $1, %bx
     1     9       9     0                             1009 test $0, %bx
     1     9       9     0                             1010 jgt .top
     1     9       0     1    ------ Interrupt ------  ------ Interrupt ------
     1     9       9     1    1004 mov  count, %ax
     1     9      10     1    1005 add  $1, %ax
     1    10      10     1    1006 mov  %ax, count
     1    10       9     0    ------ Interrupt ------  ------ Interrupt ------
     1    10       9     0                             1011 halt
     1    10      10     1    ----- Halt;Switch -----  ----- Halt;Switch -----
     0    10      10     1    1007 mov  $0, mutex
     0    10      10     0    1008 sub  $1, %bx
     0    10      10     0    1009 test $0, %bx
     0    10      10     0    1010 jgt .top
     0    10      10     0    1011 halt
```

分析结果可以看出 当一个线程执行xchg指令之后 即使被中断 另一个线程也不会同时获取到锁

下面的题目只有英文原版书中有 所以进行了题目翻译

30.1

我们的第一个问题集中在 main-two-cvs-while.c（有效的解决方案）上。 首先，研究代码。 你认为你了解当你运行程序时会发生什么吗？

生产者尝试获取锁m

生产者查看缓冲区是否已满 如果已满则等待empty信号

否则向缓冲区写入数据 发送fill信号 并释放锁继续循环

消费者尝试获取锁m

消费者查看缓冲区是否已空 如果为空则进入睡眠等待fill信号

否则从缓冲区读取数据 发送empty信号 并释放锁继续循环

30.2

指定一个生产者和一个消费者运行，并让生产者产生一些元素。 缓冲区大小从 1 开始，然后增加。随着缓冲区大小增加，程序运行结果如何改变？ 当使用不同的缓冲区大小(例如 -m 10)，生产者生产不同的产品数量(例如 -l 100)， 修改消费者的睡眠字符串(例如 -C 0,0,0,0,0,0,1)，full_num 的值如何变化？

执行以下命令

```
./main-two-cvs-while -l 3 -m 1 -p 1 -c 1 -v
./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v
./main-two-cvs-while -l 3 -m 3 -p 1 -c 1 -v
./main-two-cvs-while -l 3 -m 4 -p 1 -c 1 -v

./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v
./main-two-cvs-while -l 6 -m 2 -p 1 -c 1 -v
./main-two-cvs-while -l 12 -m 2 -p 1 -c 1 -v
./main-two-cvs-while -l 24 -m 2 -p 1 -c 1 -v

./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v -C 0,0,0,0,0,0,1
./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v -C 1,0,2,0,0,0,1
./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v -C 0,1,0,0,0,0,1
./main-two-cvs-while -l 3 -m 2 -p 1 -c 1 -v -C 0,0,2,0,0,0,1
```

只有添加缓冲区大小，它不会变化很大。
在所有测试中缓冲区都会满

在改变-C参数后 full_num的值随之变化 消费者睡眠时间越长 full_num变为2的速度越快

30.4

我们来看一些 timings。 对于一个生产者，三个消费者，大小为 1 的共享缓冲区以及每个消费者在 c3 点暂停一秒，您认需要执行多长时间？ （./main-two-cvs-while -p 1 -c 3 -m 1 -C 0,0,0,1,0,0,0:0,0,0,1,0,0,0:0,0,0,1,0,0,0 -l 10 -v -t）

指定 3 个消费者，1 个生产者，缓冲区大小为 1，
消费者睡眠点：c3、c3、c3，睡眠时间为 1 秒
每个生产者循环 10 次

如果消费者线程先执行，那么睡眠时间为 13s，
如果生产者者线程先执行，那么睡眠时间为 12s

实际结果：
Total time: 12.01 seconds
Total time: 13.01 seconds

30.8-1

现在让我们看一下 main-one-cv-while.c。您是否可以假设只有一个生产者， 一个消费者和一个大小为 1 的缓冲区，配置一个睡眠字符串，让代码运行出现问题

一个生产者和一个消费者不会出现问题

31.1

第一个问题就是实现和测试 fork/join 问题的解决方案，如本文所述。 即使在文本中描述了此解决方案， 重新自己实现一遍也是值得的。 even Bach would rewrite Vivaldi， allowing one soon-to-be master to learn from an existing one。 有关详细信息，请参见 fork-join.c。 将添加 sleep(1) 到 child 函数内以确保其正常工作。

编写代码如下

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include "common_threads.h"
#include <semaphore.h>
sem_t s;

void *child(void *arg) {
    printf("child\n");
    // use semaphore here
    sem_post(&s);
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p;
    printf("parent: begin\n");
    // init semaphore here
    sem_init(&s, 0, 0);
    Pthread_create(&p, NULL, child, NULL);
    // use semaphore here
    sem_wait(&s);
    printf("parent: end\n");
    return 0;
}
```

代码说明：

- 首先初始化信号量s 初值为0
- 在创建线程后等待信号量s
- 在线程函数中首先sleep(1) 然后打印后post信号量s
  运行结果如下

```
-> % gcc fork-join.c -lpthread && ./a.out
parent: begin
child
parent: end
```

31.2

现在，我们通过研究集合点问题 rendezvous problem 来对此进行概括。 问题如下：您有两个线程，每个线程将要在代码中进入集合点。 任何一方都不应在另一方进入之前退出代码的这一部分。 该任务使用两个信号量，有关详细信息，请参见 rendezvous.c。

编写代码如下

```
#include <stdio.h>
#include <unistd.h>
#include "common_threads.h"
#include <semaphore.h>

// If done correctly, each child should print their "before" message
// before either prints their "after" message. Test by adding sleep(1)
// calls in various locations.

sem_t s1, s2;

void *child_1(void *arg) {
    printf("child 1: before\n");
    // what goes here?
    sem_post(&s1);
    sem_wait(&s2);
    printf("child 1: after\n");
    return NULL;
}

void *child_2(void *arg) {
    printf("child 2: before\n");
    // what goes here?
    sem_post(&s2);
    sem_wait(&s1);
    printf("child 2: after\n");
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p1, p2;
    printf("parent: begin\n");
    // init semaphores here
    sem_init(&s1, 0, 0);
    sem_init(&s2, 0, 0);
    Pthread_create(&p1, NULL, child_1, NULL);
    Pthread_create(&p2, NULL, child_2, NULL);
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("parent: end\n");
    return 0;
}
```

代码说明

- 首先初始化信号量s1 s2
- 在线程0打印before后post信号量s1并等待信号s2
- 在线程1打印before后post信号量s2并等待信号s1

```
-> % gcc rendezvous.c -lpthread && ./a.out
parent: begin
child 1: before
child 2: before
child 2: after
child 1: after
parent: end
```

31.4

现在按照文本中所述，解决读者写者问题。 首先，不用考虑进程饥饿。 有关详细信息，请参见 reader-writer.c 中的代码。 将 sleep（）调用添加到您的代码中，以证明它可以按预期工作。 你能证明饥饿问题的存在吗？

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "common_threads.h"
#include <semaphore.h>

//
// Your code goes in the structure and functions below
//

typedef struct __rwlock_t {
    sem_t lock;
    sem_t write_lock;
    int reader_number;
} rwlock_t;
```

```c
void rwlock_init(rwlock_t *rw) {
    sem_init(&rw->lock, 0, 1);
    sem_init(&rw->write_lock, 0, 1);
    rw->reader_number = 0;
}

void rwlock_acquire_readlock(rwlock_t *rw) {
    sleep(1);
    sem_wait(&rw->lock);
    rw->reader_number++;
    if (rw->reader_number == 1) {
        sem_wait(&rw->write_lock);
    }
    sem_post(&rw->lock);
}

void rwlock_release_readlock(rwlock_t *rw) {
    sem_wait(&rw->lock);
    rw->reader_number--;
    if (rw->reader_number == 0) {
        sem_post(&rw->write_lock);
    }
    sem_post(&rw->lock);
}

void rwlock_acquire_writelock(rwlock_t *rw) {
    sleep(1);
    sem_wait(&rw->write_lock);
}

void rwlock_release_writelock(rwlock_t *rw) {
    sem_post(&rw->write_lock);
}

//
// Don't change the code below (just use it!)
//

int loops;
int value = 0;

rwlock_t lock;

void *reader(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
    rwlock_acquire_readlock(&lock);
    printf("read %d\n", value);
    rwlock_release_readlock(&lock);
    }
    return NULL;
}

void *writer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
    rwlock_acquire_writelock(&lock);
    value++;
    printf("write %d\n", value);
    rwlock_release_writelock(&lock);
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    assert(argc == 4);
    int num_readers = atoi(argv[1]);
    int num_writers = atoi(argv[2]);
    loops = atoi(argv[3]);

    pthread_t pr[num_readers], pw[num_writers];

    rwlock_init(&lock);

    printf("begin\n");

    int i;
    for (i = 0; i < num_readers; i++)
    Pthread_create(&pr[i], NULL, reader, NULL);
    for (i = 0; i < num_writers; i++)
    Pthread_create(&pw[i], NULL, writer, NULL);

    for (i = 0; i < num_readers; i++)
    Pthread_join(pr[i], NULL);
    for (i = 0; i < num_writers; i++)
    Pthread_join(pw[i], NULL);

    printf("end: value %d\n", value);

    return 0;
}
```

代码说明：

- 设置两个信号量 一个作为锁 另一个作为写锁 设置一个整型变量作为读者数量
- 获取读锁reader_number加1 释放读锁减1
- 当读者为0(当前没有读者 可能有写者)时获取读锁需要等待写锁释放
- 当读者为1时释放读锁(没有读者进程 可以允许写者进程写数据)post写信号
- 获取写锁等待写锁信号 释放写锁post写锁

```
-> % gcc reader-writer.c -lpthread && ./a.out 5 1 10
begin
read 0
read 0
read 0
read 0
read 0
write 1
read 1
read 1
read 1
read 1
read 1
write 2
read 2
read 2
read 2
read 2
read 2
write 3
read 3
read 3
read 3
read 3
read 3
write 4
read 4
read 4
read 4
read 4
read 4
write 5
read 5
read 5
read 5
read 5
read 5
write 6
read 6
read 6
read 6
read 6
read 6
write 7
read 7
read 7
read 7
read 7
read 7
write 8
read 8
read 8
read 8
read 8
read 8
write 9
read 9
read 9
read 9
read 9
read 9
write 10
end: value 10
```

存在的饥饿问题

只要有读者读数据 写者就无法进行操作 这会导致写者操作被长期挂起，无法操作——写者饥饿

31.5

让我们再次看一下读者写者问题，但这一次需要考虑进程饥饿。 您如何确保所有读者和写者运行？ 有关详细信息，请参见 reader-writer-nostarve.c。

在上一个问题的基础上增加了一个信号量 write_waiting

当获取读锁和获取写锁时都会先wait write_waiting信号量 这保证了当写者进程想要往里写入数据时，只允许当前存在的读者继续运行，直到结束，期间不允许其他读者到来

因为获取写锁的sem_wait会将 write_waiting的值减为0 这样其他读者再次获取读锁的时候就会被阻拦

直到写进程写完数据后 再次post write_waiting信号量的时候其他读进程才能再次获得读锁

完整代码如下

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "common_threads.h"
#include <semaphore.h>

//
// Your code goes in the structure and functions below
//

typedef struct __rwlock_t {
    sem_t lock;
    sem_t write_lock;
    sem_t write_waiting;
    int reader_number;
} rwlock_t;


void rwlock_init(rwlock_t *rw) {
    sem_init(&rw->lock, 0, 1);
    sem_init(&rw->write_lock, 0, 1);
    sem_init(&rw->write_waiting, 0, 1);
    rw->reader_number = 0;
}

void rwlock_acquire_readlock(rwlock_t *rw) {
    sleep(1);
    sem_wait(&rw->write_waiting);
    sem_wait(&rw->lock);
    rw->reader_number++;

    if (rw->reader_number == 1) {
        sem_wait(&rw->write_lock);
    }
    sem_post(&rw->lock);
    sem_post(&rw->write_waiting);

}

void rwlock_release_readlock(rwlock_t *rw) {
    sem_wait(&rw->lock);
    rw->reader_number--;
    if (rw->reader_number == 0) {
        sem_post(&rw->write_lock);
    }

    sem_post(&rw->lock);
}

void rwlock_acquire_writelock(rwlock_t *rw) {
    sleep(1);
    sem_wait(&rw->write_waiting);
    sem_wait(&rw->write_lock);
    sem_post(&rw->write_waiting);
}

void rwlock_release_writelock(rwlock_t *rw) {
    sem_post(&rw->write_lock);
}

//
// Don't change the code below (just use it!)
//

int loops;
int value = 0;

rwlock_t lock;

void *reader(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
    rwlock_acquire_readlock(&lock);
    printf("read %d\n", value);
    rwlock_release_readlock(&lock);
    }
    return NULL;
}

void *writer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
    rwlock_acquire_writelock(&lock);
    value++;
    printf("write %d\n", value);
    rwlock_release_writelock(&lock);
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    assert(argc == 4);
    int num_readers = atoi(argv[1]);
    int num_writers = atoi(argv[2]);
    loops = atoi(argv[3]);

    pthread_t pr[num_readers], pw[num_writers];

    rwlock_init(&lock);
    printf("begin\n");

    int i;
    for (i = 0; i < num_readers; i++)
    Pthread_create(&pr[i], NULL, reader, NULL);
    for (i = 0; i < num_writers; i++)
    Pthread_create(&pw[i], NULL, writer, NULL);

    for (i = 0; i < num_readers; i++)
    Pthread_join(pr[i], NULL);
    for (i = 0; i < num_writers; i++)
    Pthread_join(pw[i], NULL);

    printf("end: value %d\n", value);

    return 0;
}
```

测试结果如下：

```
-> % gcc reader-writer-nostarve.c -lpthread && ./a.out 4 2 10
begin
read 0
read 0
read 0
write 1
write 2
read 2
read 2
read 2
read 2
write 3
read 3
write 4
read 4
write 5
read 5
write 6
read 6
read 6
read 6
write 7
read 7
write 8
read 8
read 8
read 8
write 9
read 9
write 10
read 10
read 10
write 11
read 11
read 11
read 11
write 12
```

```
read 12
read 12
read 12
write 13
read 13
write 14
read 14
read 14
read 14
write 15
write 16
read 16
read 16
write 17
read 17
write 18
read 18
read 18
read 18
write 19
write 20
read 20
read 20
read 20
read 20
end: value 20
```

31.6

使用信号量构建一个没有饥饿的互斥量，其中任何试图获取该互斥量的线程都将最终获得它。 有关更多信息，请参见 mutex-nostarve.c 中的代码。

编写代码如下：

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include "common_threads.h"
#include <semaphore.h>

//
// Here, you have to write (almost) ALL the code. Oh no!
// How can you show that a thread does not starve
// when attempting to acquire this mutex you build?
//

typedef struct __ns_mutex_t {
    int turn;
    int count;
    sem_t lock;
    sem_t turnlock;
} ns_mutex_t;
```

```c
ns_mutex_t m;

void ns_mutex_init(ns_mutex_t *m) {
    Sem_init(&m->lock, 1);
    Sem_init(&m->turnlock, 1);
    m->turn = 1;
    m->count = 0;
}

void ns_mutex_acquire(ns_mutex_t *m) {
    Sem_wait(&m->turnlock);
    int turn = ++m->count;
    while (turn != m->turn) {
        Sem_post(&m->turnlock);
        Sem_wait(&m->lock);
        Sem_wait(&m->turnlock);
    }
    Sem_post(&m->turnlock);
}

void ns_mutex_release(ns_mutex_t *m) {
    Sem_wait(&m->turnlock);
    m->turn++;
    Sem_post(&m->lock);
    Sem_post(&m->turnlock);
}


void *worker(void *arg) {
    printf("Started: %lu\n", pthread_self());
    ns_mutex_acquire(&m);
    printf("Got lock: %lu\n", pthread_self());
    ns_mutex_release(&m);
    return NULL;
}

int main(int argc, char *argv[]) {
    printf("parent: begin\n");
    ns_mutex_init(&m);
    pthread_t threads[10];
    for(int i = 0; i < 10; i++) {
        Pthread_create(&threads[i], NULL, worker, NULL);
    }
    for(int i = 0; i < 10; i++) {
        Pthread_join(threads[i], NULL);
    }
    printf("parent: end\n");
    return 0;
}
```

使用一个lock信号量作为互斥的锁变量

通过在释放锁时递增turn变量 保证每个线程都会最终获得锁

测试结果如下：

```
s-> % gcc mutex-nostarve.c -lpthread && ./a.out
parent: begin
```

```
Started: 140541027841600
Got lock: 140541027841600
Started: 140541019448896
Got lock: 140541019448896
Started: 140541011056192
Got lock: 140541011056192
Started: 140541002663488
Got lock: 140541002663488
Started: 140540994270784
Got lock: 140540994270784
Started: 140540985878080
Got lock: 140540985878080
Started: 140540977485376
Got lock: 140540977485376
Started: 140540969092672
Got lock: 140540969092672
Started: 140540960699968
Got lock: 140540960699968
Started: 140540952307264
Got lock: 140540952307264
parent: end
```