第二十八章-锁

预备知识

模拟器模仿了由多个线程执行的简短汇编序列。
请注意，模拟器不会显示将要运行的 OS 代码（例如，执行上下文切换）；
因此，您所看到的只是用户代码的交叉执行。

运行的汇编代码基于 x86，但有所简化。 在此指令集中，
有四个通用寄存器（ %ax，%bx，%cx，%dx ），一个程序计数器（PC）和一小部分指令就足以满足我们的要求。

这是模拟器的所有选项，可使用-h 查看。

```
Usage: x86.py [options]

Options:
  -h, --help            显示帮助信息
  -s SEED, --seed=SEED  随机种子
  -t NUMTHREADS, --threads=NUMTHREADS
                        线程数
  -p PROGFILE, --program=PROGFILE
                        源程序 (in .s)
  -i INTFREQ, --interrupt=INTFREQ
                        中断周期
  -r, --randints        中断周期是否随机
  -a ARGV, --argv=ARGV  逗号分隔每个线程参数(例如: ax=1,ax=2 设置线程0 ax 寄存器为1,线程
1 ax 寄存器为2)
                        通过冒号分隔列表为每个线程指定多个寄存器(例如，ax=1:bx=2,cx=3设
置线程0 ax和bx，对于线程1只设置cx)
  -L LOADADDR, --loadaddr=LOADADDR
                        加载代码的地址
  -m MEMSIZE, --memsize=MEMSIZE
                        地址空间大小(KB)
  -M MEMTRACE, --memtrace=MEMTRACE
                        以逗号分隔的要跟踪的地址列表 (例如:20000,20001)
  -R REGTRACE, --regtrace=REGTRACE
                        以逗号分隔的要跟踪的寄存器列表 (例如:ax,bx,cx,dx)
  -C, --cctrace         是否跟踪条件代码condition codes)
  -S, --printstats      打印额外信息
  -c, --compute         计算结果
```

大多数参数是很容易理解的的。 使用-r 会打开一个随机周期中断器（从-i 指定为 1 到中断周期），这可以在家庭作业出现问题时带来更多乐趣。

-L 指定在地址空间的何处加载代码。

-m 指定地址空间的大小（以KB为单位）。

-S 打印额外信息

Problem1

问题描述

首先用标志 `-p flag.s` 运行 x86.py。该代码通过一个内存标志"实现"锁。你能理解汇编代码试图做什么吗?

问题解答

```
❯ cat flag.s
.var flag
.var count

.main
.top

.acquire
mov  flag, %ax      #
test $0, %ax        #
jne  .acquire       # 如果flag !=0,则跳转到.acquire处,反复检测flag是否为0
mov  $1, flag       # 获取锁(将flag设为1)

# critical section
mov  count, %ax     #
add  $1, %ax        #
mov  %ax, count     # count++

# release lock
mov  $0, flag       # 释放锁(flag设为0)

# see if we're still looping
sub  $1, %bx
test $0, %bx
jgt .top        # 如果bx的值大于0则循环(回到.top处执行)

halt
```

flag.s 作用见上面的注释，这个简单的锁有一个问题，导致它并不能保证互斥。
比如线程0执行 mov flag, %ax 完后，时钟中断，切到线程1执行，而线程1在执行 mov %ax,count 中断，切到线程0，此时线程1是拥有锁的，线程0继续执行 test $0, %ax ,这时ax的值是0，因为线程有单独的寄存器，所以线程0也获得了锁。

Problem2

问题描述

使用默认值运行时, `flag.s` 是否按预期工作?

它会产生正确的结果吗?使用-M 和-R 标志跟踪变量和寄存器(并使用-c 查看它们的值)。你能预测代码运行时标志会变成什么值吗?

问题解答

运行结果:

```
❯ python2 x86.py -p flag.s -M flag,count -c
```

```
 flag count          Thread 0                Thread 1

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

flag 最终为 0。

Problem3

问题描述

使用-a 标志更改寄存器%bx 的值(例如,如果只运行两个线程,就用-a bx=2,bx=2)。代码是做什么的?对这段代码问上面的问题,答案如何?

问题解答

count 增加 4 次(每个线程增加 2 次),flag 依然为 0,运行结果:

```
❯ python2 x86.py -p flag.s -a bx=2,bx=2 -M flag,count -c

 flag count          Thread 0                Thread 1

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

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

Problem4

问题描述

对每个线程将 bx 设置为高值,然后使用-i 标志生成不同的中断频率。什么值导致产生不好的结果?什么值导致产生良好的结果?

问题解答

见问题 1 答案的分析,
坏的情况:

```
❯ python2 x86.py -p flag.s -a bx=2,bx=2 -M flag,count -c -i 1
...
```

Problem5

问题描述

现在让我们看看程序 test-and-set.s。首先尝试理解使用 xchg 指令构建简单锁原语的代码。获取锁怎么写?释放锁如何写?

问题解答

```
❯ cat test-and-set.s
.var mutex
.var count

.main
.top

.acquire
mov  $1, %ax
xchg %ax, mutex     # 原子操作:交换ax寄存器与内存mutex空间的值(mutex设为1)
test $0, %ax        #
jne  .acquire       # 如果(%ax)!=0则自旋等待,即原mutex值不为0

# critical section
mov  count, %ax     #
add  $1, %ax        #
mov  %ax, count     # count地址的值+1

# release lock
mov  $0, mutex #  mutex设为0(释放锁)

# see if we're still looping
sub  $1, %bx
test $0, %bx  # 多次循环,直到bx值小于等于0
jgt .top

halt
```

当一个线程获取锁之后 mutex 变为 1,释放锁之后 mutex 变为 0,且操作为原子操作,解决的前面的方案带来的问题

获取锁:

```
mov  $1, %ax
xchg %ax, mutex
```

释放锁:

```
mov  $0, mutex
```

Problem6

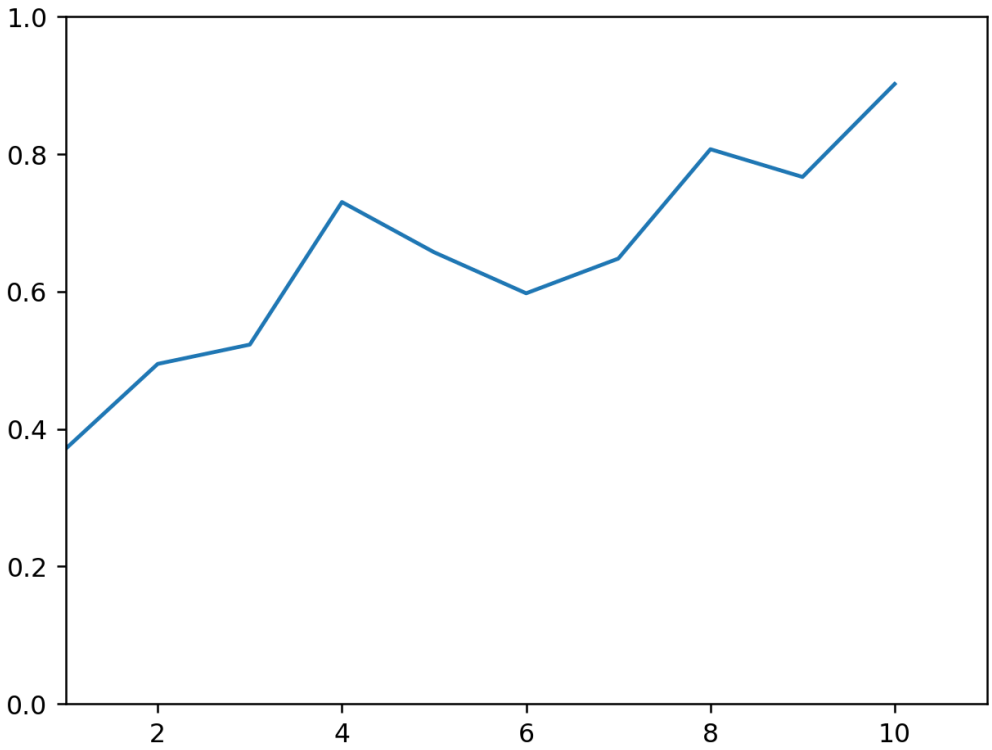
问题描述

现在运行代码,再次更改中断间隔(-i)的值,并确保循环多次。代码是否总能按预期工作?有时会导致 CPU 使用率不高吗?如何量化呢?

问题解答

```
python2 x86.py -p test-and-set.s -a bx=2,bx=2 -M count -c -i 1
python2 x86.py -p test-and-set.s -a bx=2,bx=2 -M count -c -i 2
python2 x86.py -p test-and-set.s -a bx=2,bx=2 -M count -c -i 3
```

是， 单核cpu情况下,当一个线程持有锁进入临界区时被抢占,抢占的线程将会自旋一个时间片，导致cpu利用率不高，
量化:计算 当一个线程持有锁进入临界区时被抢占,抢占线程的自旋时间长与总时间长百分比即可
具体实现: `Python3 5.py`， 显示图像,x轴为中断周期,y轴为cpu利用率



Problem7

问题描述

使用-P 标志生成锁相关代码的特定测试。例如,执行一个测试计划,在第一个线程中获取锁,但随后尝试在第二个线程中获取锁。正确的事情发生了吗?你还应该测试什么?

问题解答

```
❯ python2 x86.py -p test-and-set.s -M mutex,count -R ax,bx -c -a bx=2,bx=2 -P
0011111

mutex count      ax    bx          Thread 0                Thread 1

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

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

结果正确,测试公平性与性能(见书如何评价锁的部分)