【OS】Day30(3)

Class	Operating System: Three Easy Pieces
Date	@February 1, 2022

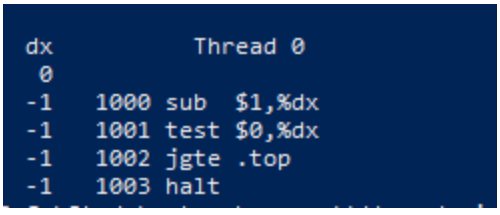
【Ch26】Concurrency: Introduction Homework

Question 1

1. Let's examine a simple program, "loop.s". First, just read and understand it. Then, run it with these arguments (`./x86.py -p loop.s -t 1 -i 100 -R dx`) This specifies a single thread, an interrupt every 100 instructions, and tracing of register `%dx`. What will `%dx` be during the run? Use the `-c` flag to check your answers; the answers, on the left, show the value of the register (or memory value) *after* the instruction on the right has run.

loop.s:

```
.main
.top
;Loop dx + 1 times
sub  $1,%dx
test $0,%dx
jgte .top
halt
```



Question 2

2. Same code, different flags: (`./x86.py -p loop.s -t 2 -i 100 -a dx=3,dx=3 -R dx`) This specifies two threads, and initializes each %dx to 3. What values will `%dx` see? Run with `-c` to check. Does the presence of multiple threads affect your calculations? Is there a race in this code?

```
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
```

The presence of multiple threads do not affect the calculation because the interrupt timer frequency enables one thread to finish before executing the next one.

There isn't a data race in the code.(

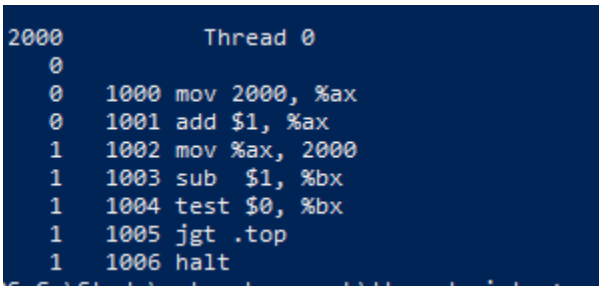
Question 3

3. Run this: `./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx` This makes the interrupt interval small/random; use different seeds (`-s`) to see different interleavings. Does the interrupt frequency change anything?

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

Question 4

4. Now, a different program, `looping-race-nolock.s`, which accesses a shared variable located at address 2000; we'll call this variable `value`. Run it with a single thread to confirm your understanding: `./x86.py -p looping-race-nolock.s -t 1 -M 2000` What is `value` (i.e., at memory address 2000) throughout the run? Use `-c` to check.



Question 5

5. Run with multiple iterations/threads: `./x86.py -p looping-race-nolock.s -t 2 -a bx=3 -M 2000` Why does each thread loop three times? What is final value of `value`?

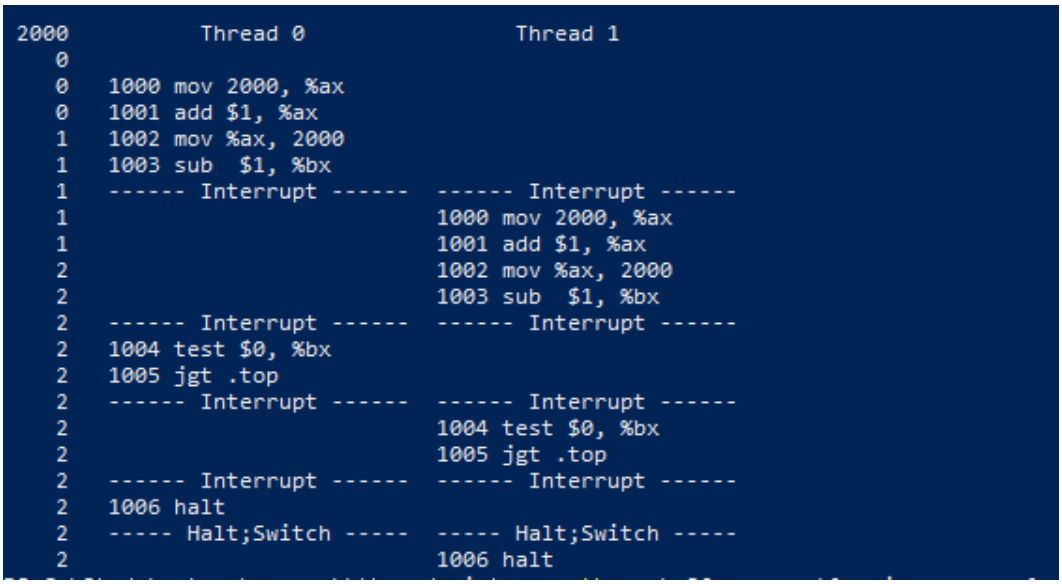
Since the starting value in bx is 3, each thread will loop 3 times. Each time it increments the value at 2000 by 1, and thus result in a final answer of 6.

```
2000          Thread 0                Thread 1
   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   1000 mov 2000, %ax
   1   1001 add $1, %ax
   2   1002 mov %ax, 2000
   2   1003 sub  $1, %bx
   2   1004 test $0, %bx
   2   1005 jgt .top
   2   1000 mov 2000, %ax
   2   1001 add $1, %ax
   3   1002 mov %ax, 2000
   3   1003 sub  $1, %bx
   3   1004 test $0, %bx
   3   1005 jgt .top
   3   1006 halt
   3   ----- Halt;Switch -----  ----- Halt;Switch -----
   3                            1000 mov 2000, %ax
   3                            1001 add $1, %ax
   4                            1002 mov %ax, 2000
   4                            1003 sub  $1, %bx
   4                            1004 test $0, %bx
   4                            1005 jgt .top
   4                            1000 mov 2000, %ax
   4                            1001 add $1, %ax
   5                            1002 mov %ax, 2000
   5                            1003 sub  $1, %bx
   5                            1004 test $0, %bx
   5                            1005 jgt .top
   5                            1000 mov 2000, %ax
   5                            1001 add $1, %ax
   6                            1002 mov %ax, 2000
   6                            1003 sub  $1, %bx
   6                            1004 test $0, %bx
   6                            1005 jgt .top
   6                            1006 halt
```

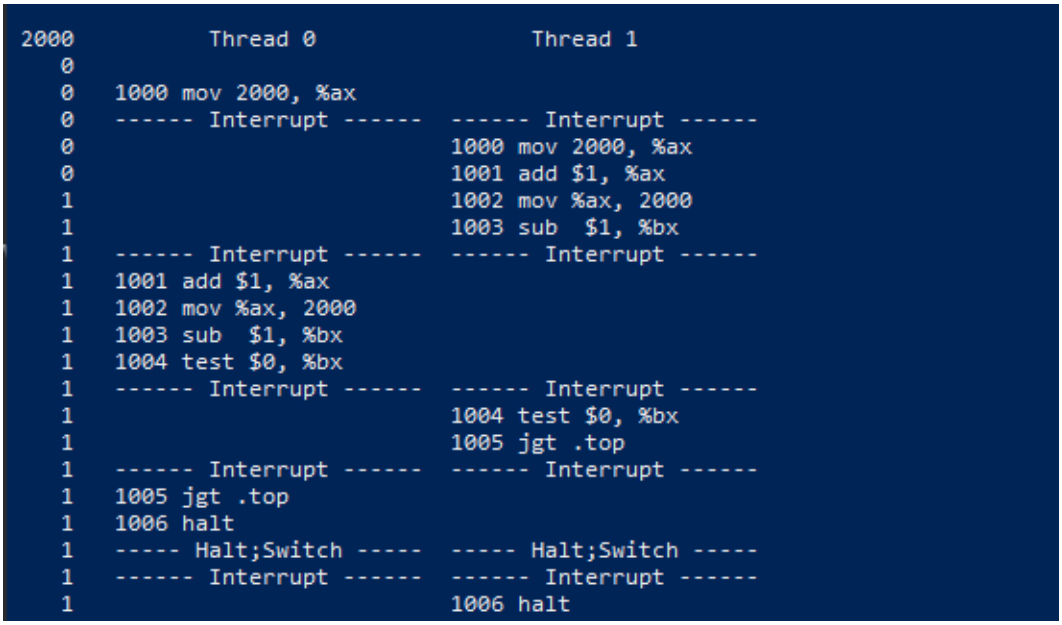
Question 6

6. Run with random interrupt intervals: `./x86.py -p looping-race-nolock.s -t 2 -M 2000 -i 4 -r -s 0` with different seeds (`-s 1`, `-s 2`, etc.) Can you tell by looking at the thread interleaving what the final value of `value` will be? Does the timing of the interrupt matter? Where can it safely occur? Where not? In other words, where is the critical section exactly?

Run with see -0:



Run with see -1:



Whether the value of ax is pushed back affects the result.

Question 7

7. Now examine fixed interrupt intervals: `./x86.py -p looping-race-nolock.s -a bx=1 -t 2 -M 2000 -i 1` What will the final value of the shared variable `value` be? What about when you change `-i 2`, `-i 3`, etc.? For which interrupt intervals does the program give the "correct" answer?

Run with i =1:

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

Run with i = 2:

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

Run with i = 3:

```
2000          Thread 0                Thread 1
   0
   0   1000 mov 2000, %ax
   0   1001 add $1, %ax
   1   1002 mov %ax, 2000
   1   ------ Interrupt ------  ------ Interrupt ------
   1                           1000 mov 2000, %ax
   1                           1001 add $1, %ax
   2                           1002 mov %ax, 2000
   2   ------ Interrupt ------  ------ Interrupt ------
   2   1003 sub  $1, %bx
   2   1004 test $0, %bx
   2   1005 jgt .top
   2   ------ Interrupt ------  ------ Interrupt ------
   2                           1003 sub  $1, %bx
   2                           1004 test $0, %bx
   2                           1005 jgt .top
   2   ------ Interrupt ------  ------ Interrupt ------
   2   1006 halt
   2   ----- Halt;Switch -----  ----- Halt;Switch -----
   2                           1006 halt
```

For i≥3 will the program give the correct answer.

Question 8

8. Run the same for more loops (e.g., set `-a bx=100`). What interrupt intervals (`-i`) lead to a correct outcome? Which intervals are surprising?

For intervals ≥ 597(execute another thread after one finishes) or interval = 3 * x, the program gives the correct answer.

Question 9

9. One last program: `wait-for-me.s`. Run: `./x86.py -p wait-for-me.s -a ax=1,ax=0 -R ax -M 2000` This sets the `%ax` register to 1 for thread 0, and 0 for thread 1, and watches `%ax` and memory location 2000. How should the code behave? How is the value at location 2000 being used by the threads? What will its final value be?

```
2000      ax          Thread 0                Thread 1
   0       1
   0       1   1000 test $1, %ax
   0       1   1001 je .signaller
   1       1   1006 mov  $1, 2000
   1       1   1007 halt
   1       0   ----- Halt;Switch -----  ----- Halt;Switch -----
   1       0                           1000 test $1, %ax
   1       0                           1001 je .signaller
   1       0                           1002 mov  2000, %cx
   1       0                           1003 test $1, %cx
   1       0                           1004 jne .waiter
   1       0                           1005 halt
```

Question 10

10. Now switch the inputs: `./x86.py -p wait-for-me.s -a ax=0,ax=1 -R ax -M 2000` How do the threads behave? What is thread 0 doing? How would changing the interrupt interval (e.g., `-i 1000`, or perhaps to use random intervals) change the trace outcome? Is the program efficiently using the CPU?

```
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       0   1002 mov  2000, %cx
  0       0   1003 test $1, %cx
  0       0   1004 jne .waiter
  0       1   ------ Interrupt ------  ------ Interrupt ------
  0       1                            1000 test $1, %ax
  0       1                            1001 je .signaller
  1       1                            1006 mov  $1, 2000
  1       1                            1007 halt
  1       0   ----- Halt;Switch -----  ----- Halt;Switch -----
  1       0   1002 mov  2000, %cx
  1       0   1003 test $1, %cx
  1       0   1004 jne .waiter
  1       0   1005 halt
```

The threads swapped. Thread 0 keeps running until thread 1 changes the value at 2000.

The CPU is not being efficiently used.