**The Threads**

**Submission:**

* Deadline: Wednesday, September 27, 2023, 8:00 pm HKT.
* Answer ALL questions. Add additional pages if necessary.
* Submit this answer sheet via Canvas->Assignments->Tutorials->Tutorial 3.

**Questions**

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.

**Answer: First, “sub” would decrement the value in %dx by 1. Then, to “test” the value in %dx (0 – 1 = -1) compare to 0. Therefore, the value in %dx is not greater than or equal to 0, so the instruction “jgte” would not jump back to the region “.top”. Finally, CPU goes straight to execute “halt”. Thus, it should be like:  
dx PC Instruction**

**-1 1000 sub $1, %dx**

**-1 1001 test $0, %dx**

**-1 1002 jgte .top**

**-1 1003 halt**

A computer screen shot of a computer program

Description automatically generated

1. 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?

**Answer: The value of %dx should starts from 3 and decrease until it becomes -1. The presence of multiple threads would not affect the calculation. Also, there is not race condition in this code since the -i flag indicates there would be interrupt after 100 instructions.**

A screen shot of a computer

Description automatically generated

1. 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 1, -s 2, etc) to see different interleavings. Does the interrupt frequency change anything?

**Answer: The interrupt frequency does change the value of %dx due to the race condition occurred in thread 0 and thread 1. Since both thread use the same register %dx, there will be data loss when there is an interrupt as known as race condition.**

A computer screen shot of a program

Description automatically generated

A screen shot of a computer

Description automatically generated

A computer screen shot of a program

Description automatically generated

1. 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.

**Answer: First, the “mov’ will copy the value of “2000” to %ax, then we “add” 1 into the value of %ax, and finally “mov’ will copy back the value in %ax to “2000”:**

1. **“2000” -> %ax**
2. **%ax + 1**
3. **%ax -> “2000”**

**We use the register %bx be the looping condition controlled variable, so that we can set how many times of the looping as we initialize the value in %bx.**

**Therefore, the value of “2000” should be 1, because the initial value of %bx is 0 and then decrement by 1 (“sub”), so that unable to satisfy the condition “jgt” (greater than 0), thus the jump instruction will not be executed and halted.**

A computer screen shot of a computer code

Description automatically generated

1. 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?

**Answer: Since we have the initial value of %bx be 3 and mentioned before that %bx is our controlled variable. Therefore, the value of %bx will decrement from 3 to 0. Hence, there will be three looping times for each thread. As result, we can expect the final value of “2000” will be 3.**

A computer screen shot of a program

Description automatically generated

1. 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?

**Answer: We don’t know the final value of having seed 1 or seed 2, since there will be -r flag as random interrupt frequency. The timing of the interrupt is matter because it may causes race condition between different thread, the critical section will be “add” $1, %ax and “mov” %ax, 2000. It is because the value of %ax or “2000” would be “erased” or loss after we changing the thread as known as interrupt.**

A computer screen shot of a program

Description automatically generated

A computer screen shot of a program

Description automatically generated

A computer screen shot of a program

Description automatically generated

1. 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?

**Answer: The final value of “2000” should be 1 for -i 1 and -i 2, and 2 for -i 3.**

**The -i 3 interrupt interval gives the “correct” answer without having any race condition.**

A screen shot of a computer

Description automatically generated

A computer screen shot of a program

Description automatically generated

A computer screen shot of a program

Description automatically generated

1. 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 (unexpected)?

**Answer: -i 3 flag leads to a correct outcome as we mentioned before that the interrupt interval is not leading any race condition. The -i 1 and -i 2 are giving unexpected value since the race condition occurred.**

A screen shot of a computer program

Description automatically generated

A computer screen shot of a program

Description automatically generated

A screenshot of a computer screen

Description automatically generated