第二十六章-并发_介绍

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

开始,我们来看一个简单的程序,“loop.s”。首先,阅读这个程序,看看你是否能理解它: `cat loop.s` 。然后,用这些参数运行它:

```
./x86.py -p loop.s -t 1 -i 100 -R dx
```

这指定了一个单线程,每 100 条指令产生一个中断,并且追踪寄存器 %dx。你能弄清楚 %dx 在运行过程中的值吗?
你有答案之后,运行上面的代码并使用 -c 标志来检查你的答案。注意答案的左边显示了右侧指令运行后寄存器的值(或内存的值)。

问题解答

loop.s 文件:

```
.main
.top          # 标号
sub  $1,%dx   # dx寄存器值减1, 结果存入dx
test $0,%dx   # 比较dx寄存器的值与0
jgte .top     # 如果dx寄存器的值大于或等于0,则跳转到标号.top处
halt          # 结束线程
```

运行:

```
python2 x86.py -p loop.s -t 1 -i 100 -R dx
```

-t 指定线程数为 1, -i 指定每个线程 100 条指令中断一次, -R 指定跟踪 dx 寄存器的值
可以添加 `-a dx=0` 参数初始化 dx 寄存器的值

运行结果:

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

```
dx          Thread 0
0                       # dx初始值为0
-1   1000 sub  $1,%dx   # dx减1后存入dx中，得到-1
-1   1001 test $0,%dx   # 比较dx和0的大小关系
-1   1002 jgte .top     # 如果dx≥0，跳转至.top处执行，否则顺序执行下一条指令
-1   1003 halt          # 执行halt结束线程
```

答案验证

```
python2 x86.py -p loop.s -t 1 -i 100 -R dx -c

...

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

经验证，答案正确。

Problem2

问题描述

现在运行相同的代码,但使用这些标志:

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

这指定了两个线程,并将每个%dx 寄存器初始化为 3。%dx 会看到什么值?使用-c 标志运行以查看答案。多个线程的存在是否会影响计算？这段代码有竞态条件吗?

问题解答

运行(线程数为 2, 每个线程 100 条指令中断一次, 线程 0,1 的 dx 寄存器都初始化为 3,跟踪 dx 寄存器的值):

```
python2 x86.py -p loop.s -t 2 -i
 100 -a dx=3,dx=3 -R dx
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
    ?
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1000 sub  $1,%dx
    ?   1001 test $0,%dx
    ?   1002 jgte .top
    ?   1003 halt
    ?   ----- Halt;Switch -----  ----- Halt;Switch -----
    ?                           1000 sub  $1,%dx
    ?                           1001 test $0,%dx
    ?                           1002 jgte .top
    ?                           1000 sub  $1,%dx
    ?                           1001 test $0,%dx
    ?                           1002 jgte .top
    ?                           1000 sub  $1,%dx
    ?                           1001 test $0,%dx
    ?                           1002 jgte .top
    ?                           1000 sub  $1,%dx
    ?                           1001 test $0,%dx
    ?                           1002 jgte .top
    ?                           1003 halt
```

```
dx          Thread 0                Thread 1
3                       # dx初始值为3
2    1000 sub  $1,%dx   # dx减1后存入dx中，得到2
2    1001 test $0,%dx   # 比较dx和0的大小关系
2    1002 jgte .top     # 如果dx≥0，跳转至.top处执行，否则顺序执行下一条指令
1    1000 sub  $1,%dx   # dx减1后存入dx中，得到1
1    1001 test $0,%dx   # 比较dx和0的大小关系
1    1002 jgte .top     # 如果dx≥0，跳转至.top处执行，否则顺序执行下一条指令
0    1000 sub  $1,%dx   # dx减1后存入dx中，得到0
0    1001 test $0,%dx   # 比较dx和0的大小关系
0    1002 jgte .top     # 如果dx≥0，跳转至.top处执行，否则顺序执行下一条指令
-1   1000 sub  $1,%dx   # dx减1后存入dx中，得到-1
-1   1001 test $0,%dx   # 比较dx和0的大小关系
-1   1002 jgte .top     # 如果dx≥0，跳转至.top处执行，否则顺序执行下一条指令
-1   1003 halt          # 执行halt结束线程
3                                           # dx初始值为3
2                           1000 sub  $1,%dx   # dx减1后存入dx中，得到2
2                           1001 test $0,%dx   # 比较dx和0的大小关系
2                           1002 jgte .top     # 如果dx≥0，跳转至.top处执行，否则
顺序执行下一条指令
1                           1000 sub  $1,%dx   # dx减1后存入dx中，得到1
1                           1001 test $0,%dx   # 比较dx和0的大小关系
1                           1002 jgte .top     # 如果dx≥0，跳转至.top处执行，否则
顺序执行下一条指令
0                           1000 sub  $1,%dx   # dx减1后存入dx中，得到0
0                           1001 test $0,%dx   # 比较dx和0的大小关系
```

```
0                                   1002 jgte .top      # 如果dx≥0，跳转至.top处执行，否则
顺序执行下一条指令
-1                                  1000 sub  $1,%dx    # dx减1后存入dx中，得到-1
-1                                  1001 test $0,%dx    # 比较dx和0的大小关系
-1                                  1002 jgte .top      # 如果dx≥0，跳转至.top处执行，否则
顺序执行下一条指令
-1                                  1003 halt           # 执行halt结束线程
```

线程 1 运行结果与线程 0 相同，多个线程不会影响计算，因为指令执行长度小于中断周期，在当前线程执行完成前不会进行线程切换。这段代码没有竞态条件。

答案验证

```
python2 x86.py -p loop.s -t 2 -i 100 -a dx=3,dx=3 -R dx -c

...

   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
```

经验证，答案正确。

Problem3

问题描述

现在运行以下命令:

```
./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx
```

这使得中断间隔非常小且随机，使用不同的种子和-s 来查看不同的交替。中断频率是否会改变这个程序的行为?

问题解答

运行(线程数为 2, 每个线程 1-3 条指令中断一次, 两个线程寄存器都初始化为 3,跟踪 dx 寄存器的值):

```
python2 x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx -c -s 0

...

   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

python2 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

python2 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
```

中断频率会改变程序的行为，当中断发生在临界区时可能会导致一些问题。在本题中，由于线程0和线程1没有共享资源，所以不会产生不确定的结果。

Problem4

问题描述

接下来我们将研究一个不同的程序（looping-race-nolock.s）。

该程序访问位于内存地址 2000 的共享变量，简单起见，我们称这个变量为 x.使用单线程运行它，并确保你了解它的功能，如下所示:

```
./x86.py -p looping-race-nolock.s -t 1 -M 2000
```

在整个运行过程中， x（即内存地址为 2000）的值是多少？使用-c 来检查你的答案。

问题解答

运行(线程数为 1):

```
python2 x86.py -p looping-race-nolock.s -t 1 -M 2000
```

结果:

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

答案:

```
 2000          Thread 0
    0                                  # x初始值为0,bx初始值为0
    0   1000 mov 2000, %ax             # %ax=0
    0   1001 add $1, %ax              # 将ax+1存入ax中，%ax=1
    1   1002 mov %ax, 2000            # 将ax写回x，x=1
    1   1003 sub  $1, %bx             # 将bx-1存入bx中，%bx=-1
    1   1004 test $0, %bx             # 比较bx和0的大小关系
    1   1005 jgt .top                 # 如果bx>0，跳转至.top处执行，否则顺序执行下一条
指令
    1   1006 halt                     # 执行halt结束线程
```

答案验证

```
python2 x86.py -p looping-race-nolock.s -t 1 -M 2000 -c

...

 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
```