黄玟瑜
19335074
huangmy73@mail2.sysu.edu.cn

作业四

操作系统原理, 2021 春

2021-04-11

任务一

完成教材习题 4.1、4.2、4.7、4.9。

习题 4.1

是的。同一个进程中的两个线程在切换时需要切换 CPU 上下文和切换内核栈，而不同进程间的线程的切换除了需要切换 CPU 上下文和切换内核栈外，还需要切换虚拟地址空间，且两个进程间的切换需要存储更多的状态信息，因此线程间的切换比进程的切换开销要少。

习题 4.2

因为对于用户级线程来说，一个进程的线程结构对操作系统来说是不可见的，而操作系统的调度是以进程为单位的，因此当用户级线程执行系统调用时操作系统阻塞其会阻塞这个进程中所有的线程。

习题 4.7

a.

该函数的功能为统计给定链表 l 中大于 0 的数据的个数，结果存储在全局变量 global_positives 中。

b.

若该函数在两个并发的线程中使用会出现问题。两个线程共享同一个变量 global_positives，若读写操作互斥，两个线程都运行该程序的结果是 global_positives 被累计了两遍，最终统计的正数的个数是链表 l 中正数的实际个数的 2 倍；若读写操作不互斥，则有可能发生 race condition，例如线程 A 读取 global_positives 时它的值为 1，加 1 后的结果为 2，若此时线程 B 执行一段时间将要写回的值为 7，线程 A 再将 2 写回，累计的结果显然是错误的。

习题 4.9

a.

该程序对变量 myglobal 累加 20 次，它创建子线程并让子线程执行和主线程相同功能的代码，但子线程每次取数存储到局部变量 j 中，对 j 操作完延时 1 后再将结果写回变量 myglobal 中，主线程直接在 myglobal 上加 1，延时 1 后再重复操作。

b.

输出结果不是所期望的。可以看到输出序列中有 21 个“.”，20 个“o”，按正常输出应该包含 20 个“.”和 20 个“o”。该程序模拟了没有读写锁的情况下不同线程的竞争，子线程取数，操作完后延时 1s，在此期间主线程完成了一次对 myglobal 的修改，之后子线程再将结果写回导致主线程的操作被覆盖了（主线程累加的数被吞了），这一过程表现为出现“.o”，可以观察到序列中出现了 19 次“.o”，说明此次程序的运行中主线程对 myglobal 的累加有 19 次无效，因此结果为 40-19=21，若将子线程程序的 sleep（1）；移动到 myglobal=myglobal+1；之后，则输出结果为 40。

任务二

运行 OSTEP 提供的参考代码：HW-ThreadsIntro.tgz，并在此基础上编写自己的汇编程序，截屏展示结果，特别展示多线程竞争的结果，相关代码可以在本课程网站和 OSTEP 课程网站下载：

https://pages.cs.wisc.edu/ remzi/OSTEP/Homework/

运行示例

运行 simple-race.s

```
D:\Python practise\practise\OS>python x86.py -p simple-race.s -t 1 -M 2000 -R ax,bx -c
ARG seed 0
ARG numthreads 1
ARG program simple-race.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG argv
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace ax,bx
ARG cctrace False
ARG printstats False
ARG verbose False

 2000      ax    bx          Thread 0
    0       0     0
    0       0     0   1000 mov 2000(%bx), %ax
    0       1     0   1001 add $1, %ax
    1       1     0   1002 mov %ax, 2000(%bx)
    1       1     0   1003 halt
```

可以看到这段程序将内存地址 2000 处的值取出，加 1 后写回。

- -p source program ，指定该程序为源程序
- -t number of threads，指定了线程数目
- -M comma-separated list of addrs to trace ，指定要 trace 的地址空间范围
- -R comma-separated list of regs to trace ，指定要 trace 的寄存器
- -c compute answers for me，计算结果，相当于给各个变量赋初始值

运行 loop.s ，设置 dx 的初始值为 3，显示 condition codes

```
D:\Python practise\practise\OS>python x86.py -p loop.s -t 1 -a dx=3 -R dx -C -c
ARG seed 0
ARG numthreads 1
ARG program loop.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG argv dx=3
ARG load address 1000
ARG memsize 128
ARG memtrace
ARG regtrace dx
ARG cctrace True
ARG printstats False
ARG verbose False

  dx   >= >  <= <  != ==        Thread 0
   3   0  0  0  0  0  0
   2   0  0  0  0  0  0  1000 sub  $1,%dx
   2   1  1  0  0  1  0  1001 test $0,%dx
   2   1  1  0  0  1  0  1002 jgte .top
   1   1  1  0  0  1  0  1000 sub  $1,%dx
   1   1  1  0  0  1  0  1001 test $0,%dx
   1   1  1  0  0  1  0  1002 jgte .top
   0   1  1  0  0  1  0  1000 sub  $1,%dx
   0   1  0  1  0  0  1  1001 test $0,%dx
   0   1  0  1  0  0  1  1002 jgte .top
  -1   1  0  1  0  0  1  1000 sub  $1,%dx
  -1   0  0  1  1  1  0  1001 test $0,%dx
  -1   0  0  1  1  1  0  1002 jgte .top
  -1   0  0  1  1  1  0  1003 halt
```

这是一个简单的循环，给寄存器 dx 赋初始值 3，每循环一次 dx 的值减 1，dx 值小于 0 时跳出循环。

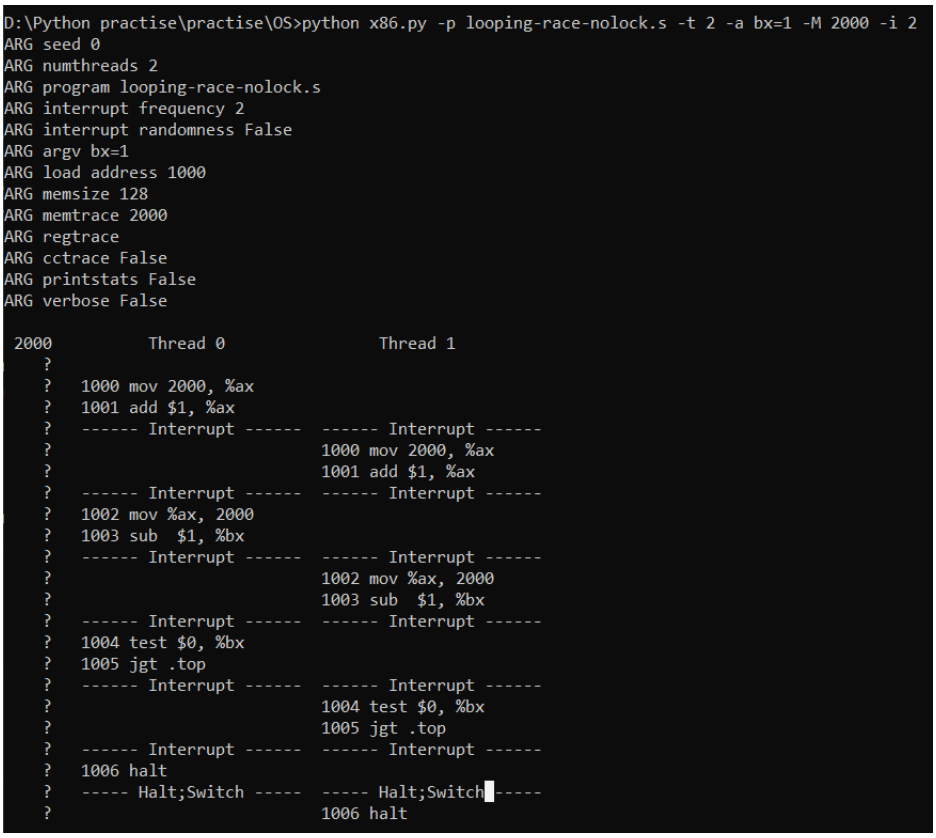
- -C should we trace condition codes，追踪标志

运行 looping-race-nolock.s，设置线程数为 2

```
D:\Python practise\practise\OS>python x86.py -p looping-race-nolock.s -t 2 -a bx=1 -M 2000 -c
ARG seed 0
ARG numthreads 2
ARG program looping-race-nolock.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG argv bx=1
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace
ARG cctrace False
ARG printstats False
ARG verbose False

 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   1006 halt
    1   ----- Halt;Switch -----  ----- Halt;Switch -----
    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                           1006 halt
```

运行 looping-race-nolock.s，设置线程数为 2，中断频率每 2 条指令一次



- -i interrupt frequency，中断频率

至此我学会了模拟器的使用，包括各种参数的设置，下面来编写自己的汇编程序。

编写程序

程序如下：

```
# assumes %bx stores the Thread ID
# e.g., Thread 0 holds bx=0, Thread 1 holds bx=1

.main
# critical section
mov 2000, %ax  # get 'value' at address 2000
add %bx, %ax    # add %bx to %ax
mov %ax, 2000  # store it back

halt
```

该程序将各个线程的线程号相加，汇总到内存地址为 2000 的值中。

若各个线程串行执行，运行 4 个线程的结果如下：

```
D:\Python practise\practise\OS>python x86.py -p race.s -t 4  -M 2000 -c -a bx=0,bx=1,bx=2,bx=3 -R ax
ARG seed 0
ARG numthreads 4
ARG program race.s
ARG interrupt frequency 50
ARG interrupt randomness False
ARG argv bx=0,bx=1,bx=2,bx=3
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace ax
ARG cctrace False
ARG printstats False
ARG verbose False

 2000      ax         Thread 0                Thread 1                Thread 2                Thread 3
    0       0
    0       0   1000 mov 2000, %ax
    0       0   1001 add %bx, %ax
    0       0   1002 mov %ax, 2000
    0       0   1003 halt
    0       0   ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----
    0       0                           1000 mov 2000, %ax
    0       1                           1001 add %bx, %ax
    1       1                           1002 mov %ax, 2000
    1       1                           1003 halt
    1       0   ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----
    1       1                                                   1000 mov 2000, %ax
    1       3                                                   1001 add %bx, %ax
    3       3                                                   1002 mov %ax, 2000
    3       3                                                   1003 halt
    3       0   ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----
    3       3                                                                               1000 mov 2000, %ax
    3       6                                                                               1001 add %bx, %ax
    6       6                                                                               1002 mov %ax, 2000
    6       6                                                                               1003 halt
```

当线程数量为 4 时，累加结果应该为 0+1+2+3=6，当线程依次执行这段程序互不影响时，结果应为 6，如图中所示。

设置 interrupt frequency=2，运行结果如下：

```
D:\Python practise\practise\OS>python x86.py -p race.s -t 4  -M 2000 -c -a bx=0,bx=1,bx=2,bx=3 -R ax -i 2
ARG seed 0
ARG numthreads 4
ARG program race.s
ARG interrupt frequency 2
ARG interrupt randomness False
ARG argv bx=0,bx=1,bx=2,bx=3
ARG load address 1000
ARG memsize 128
ARG memtrace 2000
ARG regtrace ax
ARG cctrace False
ARG printstats False
ARG verbose False

 2000      ax         Thread 0                Thread 1                Thread 2                Thread 3
    0       0
    0       0   1000 mov 2000, %ax
    0       0   1001 add %bx, %ax
    0       0   ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------
    0       0                           1000 mov 2000, %ax
    0       1                           1001 add %bx, %ax
    0       0   ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------
    0       0                                                   1000 mov 2000, %ax
    0       2                                                   1001 add %bx, %ax
    0       0   ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------
    0       0                                                                           1000 mov 2000, %ax
    0       3                                                                           1001 add %bx, %ax
    0       0   ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------
    0       0   1002 mov %ax, 2000
    0       0   1003 halt
    0       1   ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----
    0       2   ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------
    2       2                                                   1002 mov %ax, 2000
    2       2                                                   1003 halt
    2       3   ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----
    2       1   ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------
    1       1                           1002 mov %ax, 2000
    1       1                           1003 halt
    1       3   ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----  ----- Halt;Switch -----
    1       3   ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------  ------ Interrupt ------
    3       3                                                                           1002 mov %ax, 2000
    3       3                                                                           1003 halt
```

设置中断频率为 2 相当于模拟了线程并行运行时异步的状态，如图中，当 Tread 2 取数时，取到的数应为 Tread 0 和 Tread 1 的和 1，而此时 Tread 0 和 Tread 1 的运算结果都还未写回到内存中，这将导致结果错误。

线程是非独立的，同一个进程里线程是数据共享的，当各个线程访问数据资源时会出现竞争状态即：数据几乎同步会被多个线程占用，即所谓的线程不安全，最终造成数据混乱。