《操作系统》课下作业 (OS-HW3)

中国人民大学 信息学院 崔冠宇 2018202147

P278, 5.6 Consider the following processes P1 and P2 that update the value of the shared variables, x and y, as follows:

```
Process P1 :
(  performs the operations:
    x := x * y
    y ++
)
LOAD R1, X
LOAD R2, Y
MUL R1, R2
STORE X, R1
INC R2
STORE Y, R2
```

```
Process P2 :
(  performs the operations:
    x ++
    y := x * y
)
LOAD R3, X
INC R3
LOAD R4, Y
MUL R4, R3
STORE X, R3
STORE Y, R4
```

Assume that the initial values of x and y are 2 and 3 respectively. P1 enters the system first and so it is required that the output is equivalent to a serial execution of P1 followed by P2. The scheduler in the uniprocessor system implements a pseudo-parallel execution of these two concurrent processes by interleaving their instructions without restricting the order of the interleaving.

a. If the processes P1 and P2 had executed serially, what would the values of x and y have been after the execution of both processes?

b. Write an interleaved concurrent schedule that gives the same output as a serial schedule.

c. Write an interleaved concurrent schedule that gives an output that is different from that of a serial schedule.

解. (题目大意: 考虑下列更新共享变量 x 及 y 的进程 P1 和 P2:(代码略) 假定 x 和 y 的初始值分别为 2 和 3. P1 首先进入系统, 并且要求输出结果等价于 P1、P2 串行执行. 单处理器系统上的调度器通过交替执行它们的指令, 但又不严格限制它们的交替顺序, 实现了这两个并发进程的伪并行执行. **a.** 如果进程 P1 和 P2 已经串行执行完毕, 在执行完两个进程后, x 和 y 的值将会是什么? **b.** 写一种交替调度, 使得与串行调度有相同的输出. **c.** 写一种交替调度, 使得与串行调度有不相同的输出.)

a. 相当于按顺序执行下列语句:

```
int x = 2, y = 3;
x = x * y; // x == 6, y == 3
y ++; // x == 6, y == 4
x ++; // x == 7, y == 4
y = x * y; // x == 7, y == 28
```

所以执行完后, x 的值为 7, y 的值为 28.

b. 一种符合条件的调度(红字表示 P1, 蓝字表示 P2):

- `LOAD R1, X`
- `LOAD R2, Y`
- `MUL R1, R2`
- `STORE X, R1`(这一句是关键, 将运算结果复制回内存)

(等价于 `x = x * y` 操作)

- `LOAD R3, X`
- `INC R3`

(基本等价于 `x++` 操作)

- `INC R2`
- `STORE Y, R2`(这一句是关键, 将运算结果复制回内存)

(等价于 `y++` 操作)

- `LOAD R4, Y`
- `MUL R4, R3`
- `STORE X, R3`
- `STORE Y, R4`

(等价于 `y = x * y` 操作)

由于 `x++` 与 `y++` 交换顺序不产生影响, 所以值($x = 7, y = 28$)是一致的.

c.修改上面的关键语句的顺序:

- `LOAD R1, X`
- `LOAD R2, Y`
- `MUL R1, R2`

- LOAD R3, X
- INC R3
- STORE X, R1
- INC R2
- STORE Y, R2
- LOAD R4, Y
- MUL R4, R3
- STORE X, R3
- STORE Y, R4

运行结果为: x = 3, y = 12.

P287, 5.26 This problem demonstrates the use of semaphores to coordinate three types of processes. Santa Claus sleeps in his shop at the North Pole and can only be awakened by either (1) all nine reindeer being back from their vacation in the South Pacific, or (2) some of the elves having difficulties making toys; to allow Santa to get some sleep, the elves can only wake him when three of them have problems. When three elves are having their problems solved, any other elves wishing to visit Santa must wait for those elves to return. If Santa wakes up to find three elves waiting at his shop's door, along with the last reindeer having come back from the tropics, Santa has decided that the elves can wait until after Christmas, because it is more important to get his sleigh ready. (It is assumed the reindeer do not want to leave the tropics, and therefore they stay there until the last possible moment.) The last reindeer to arrive must get Santa while the others wait in a warming hut before being harnessed to the sleigh. Solve this problem using semaphores.

解. (题目大意: 本问题展示了用信号量来协调三种进程的方法. 睡在北极商店里的圣诞老人只能被下列情景唤醒: (1) 所有9头驯鹿都从南太平洋度假回来, 或是 (2) 有些精灵制作玩具时遇到了麻烦; 为了让圣诞老人多睡一会儿, 只能在3个精灵有麻烦的时候才能叫醒圣诞老人. 当3个小精灵解决问题时, 其他想要找圣诞老人的小精灵只能等那些精灵返回. 如果圣诞老人醒来后发现3个小精灵以及最后一头从热带度假归来的驯鹿在店门口等着, 那么圣诞老人就决定让这些小精灵等到圣诞节以后, 因为准备雪橇更重要.(假设驯鹿不想离开热带, 因此它们要待到最后可能的时刻.) 最后的驯鹿一定要赶回来找圣诞老人, 在套上雪橇之前, 驯鹿会在温暖的棚子里等着. 用信号量解决这个问题.)

大致思路如下所示:

```
// Wait 9 deer.
#define DEERMAX 9
```

```
// Wait 3 elves.
#define ELVESMAX 3

// Is Santa awake?
semaphore callSanta = 0;
// Is sleigh prepared?
semaphore sleigh = 0;
// Deer waits in hut if comes back early.
semaphore deerWait = 0;
// Is 3 elves have problem?
semaphore enoughElves = 0;
// Elf waits for Santa if santa is out.
semaphore santaInHouse = 1;

// Deer ready count.
int deerBack = 0;
mutex m_deerBack = 1;
// Elf with problem count.
int elfProblem = 0;
mutex m_elfProblem = 1;

// ----Process deer.----
while(1)
{
    takeVacation();
    // Go back.
    goBack();
    semWait(m_deerBack);
    deerBack++;
    semSignal(m_deerBack);
    // All 9 deer come back.
    if(deerBack == DEERMAX)
    {
        semSignal(callSanta);
    }
    // Or wait in hut.
```

```
    semWait(deerWait);
    semWait(sleigh);
    fly();
}
// ----Process elf. ----
while(1)
{
    bool hasProblem = makingGifts();
    // Has problem.
    if(hasProblem)
    {
        semWait(m_elfProblem);
        elfProblem++;
        semSignal(m_elfProblem);
        // 3 elves have problem.
        if(elfProblem == ELVESMAX)
        {
            semSignal(enoughElves);
        }
        // Wait 3 elves.
        semWait(enoughElves);
        // Wait santa come back.
        semWait(santaInHouse);
        // Call santa.
        semSignal(callSanta);
        askQuestion();
    }
}
// ----Process santa.----
while(1)
{
    // Wait until something happens.
    semWait(callSanta);
    // All deer come back.
    if(deerReady == DEERMAX)
    {
```

```
        for(int i = 0; i < DEERMAX; i++)
        {
            // Stop deer wait, and prepare sleigh.
            semSignal(deerWait);
            prepareSleigh();
            semSignal(sleigh);
        }
        deliverGifts();
        deerBack = 0;
    }
    // 3 elves come.
    else
    {
        semWait(m_elfProblem);
        elfProblem -= ELVESMAX;
        semSignal(m_elfProblem);
        solveProblem();
        semSignal(santaInHouse);
    }
}
```

P288, 5.28 Explain what is the problem with this implementation of the one-writer many-readers problem?

```
int readcount;                  // shared and initialized to 0
Semaphore mutex, wrt;           // shared and initialized to 1;
// Writer :                     // Readers :
                                semWait(mutex);
                                readcount := readcount + 1;
semWait(wrt);                   if readcount == 1 then semWait(wrt);
/* Writing performed*/          semSignal(mutex);
semSignal(wrt);                 /*reading performed*/
                                semWait(mutex);
                                readcount := readcount - 1;
                                if readcount == 0 then Up(wrt);
                                semSignal(mutex);
```

解. (题目大意: 解释下面这种单写者多读者问题的实现的问题出在哪里?)

如若假定写者进程总是先于读者进程, 是没有问题的; 但是如果相反, 读者进程可能会没有数据可读. 而且我们也无法保证临界区内仅有一个读者进程, 可能总有读者进程保持在临界区内, `Up(wrt)` 一直得不到执行, 这样会导致写者进程一直无法执行.