Universitetet i Oslo
Institutt for Informatikk
PMA
Olaf Owe, Martin Steffen

INF 4140: Models of Concurrency

Høst 2015 **Series 1** 24. 08. 2015

Topic: Warm-up: thinking concurrently and basic synchronization

Issued: 24. 08. 2015

Exercise 1 (Parallelism and concurrency) The notions of *parallelism* and *concurrency*, while related, are not identical.[1] Parallelism implies that executions "really" run at the same physical time, whereas concurrent execution may happen on a mono-processor, where the fact that various processes seem to happen simultaneously is just an "illusion" (typically an illusion maintained by the operating system).

Assume you have a mono-processor (and a single-core machine), so the CPU does not contain parallel hardware. Under these circumstances, does it make sense to use concurrency when programming? Is it possible, that using concurrency makes programs run faster? Give reason for your opinion.

```
For the exercises: do exercises 2.1, 2.2, 2.10, 2.12, 2.13, 2.14,
and 2.15  from the textbook (it's not forbidden to have a look/spend a
thought at the others as well ...)
```

Exercise 2 (Synchronization) Do the Exercise 2.1 from the book. Consider the skeleton of a program in Listing 1.

1. add missing synchronization code, to access the `buffer` appropriately.
2. do the same for a situation, when there are *two files* to read from. The reading should be done "independently", i.e., you need 2 processes, each one reads from its own file.

Listing 1: Finding pattern in a file (skeleton)

```
1 string buffer              # contains 1 line of the input
2 bool   done = false;
3 co
4     string line1;
5     while (true) {
6       wait for buffer to be full or done to be true;
7       if done break;
8       line1 := buffer;
```

[1] The terminology is not 100% uniform across all fields. Nonetheless, the one we use in the lecture is the most common one.

```
9           signal that buffer is empty;
10          look for patterns in line1;
11          if (pattern is in line1)
12              write line1;
13      }
14  ||
15
16      string line2;
17      while (true) {
18          read next line of input into line2;
19          if EOF { done := true; break};
20          wait for buffer to be empty;
21          buffer := line2;
22          signal that buffer is full;
23      }
24  co;
```

Remark: This is a version of the so-called, well-known, practically important and ubiquitous "readers-writers-problem". Be careful with the words: the process(es) that read from the *file* in this example here are the *writers* of the readers/writers-problem (as they *write* to the "memory" *shared between readers and writers*, and consequently the process that writes the lines (after finding a pattern) is a "reader" process in the readers-writers problem.

Exercise 3 Consider the code of the simple producer-consumer problem. Change it so that the variables p is process-local, not global.

Listing 2: Producer/consumer, global `p`

```
1
2                   int buf, p := 0; c := 0;
3
4   process Producer {                  process Consumer {
5     int a[N];...                      int b[N];...
6     while (p < N) {                   while (c < N) {
7       < await (p = c) ; >              < await (p > c) ; >
8       buf := a[p];                      b[c] := buf;
9         p := p+1;                          c := c+1;
10    }                                   }
11  }                                   }
```

Exercise 4 (Histories and atomicity) Do Exercise 2.10 from the book. Consider the shown code. How many histories are there? What are they possible final values. Compare the situations where the assignments are considered atomic, and where they are not?

```
1   int x = 0, y = 0;
2   co
3       x := x + 1; # S1
4       x := x + 2; # S2
5   ||
6       x := x + 2; # P1
7       y := y - x; # P2
8   oc
```

Exercise 5 (Interleaving, non-determinism, and atomicity) Do Exercise 2.12 from the book. Consider

```
1   int x = 2, y = 3;
2
3   co
4     <x := x + y;> #S1
5   ||
6     <y := x * y;>   #S2
7   oc
```

1. does the prog. satisfies the AMO property?
2. what's the result(s)?

Exercise 6 Do Exercise 2.13 from the book.

```
1   S1 =  x := x+2        S2 =  y := x-y          S3 =  x := x-y
2
3   init:  x=2, y=5
4   ----------------------
5
6   a) S1; S2; S3
7
8   b) co
9         <S1;>
10      ||
11        <S2;>
12      || <S3;>
13    oc
14
15  c) co <await x > y S1;S2> || S3 oc
```

Exercise 7 (At most once) Do Exercise 2.14 from the book. Consider the following code.

1. does it satisfy AMO?
2. What are the final values for x and y? Explain your answer.

```
1   int x = 1, y = 1;
2   co
3     <x := x+y;>       #1
4   ||
5     y := 0;           #2
6   ||
7     x := x - y;       #3
8   oc
```

Exercise 8 (AMO, termination) Do exercise 2.15 from the book.

```
1   int x = 0, y = 10;
2
3   co
```

```
4       while (x != y) x := x + 1;
5   ||
6       while (x != y) y := y - 1;
7   oc
```

1. AMO?
2. Termination?