Universitetet i Oslo
Institutt for Informatikk

Research group for Reliable Systems (PSY)
Andrea Pferscher

INF 5170: Models of Concurrency

Fall 2025

Group Session 1

29.08.2025

Topic: Warm-up: thinking concurrently and basic synchronization

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 which requires a multi-core CPU. Concurrent execution may happen on a single-core 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 single-core processor, so the CPU does not contain parallel hardware. Under these circumstances, is it possible, that using concurrency makes programs run faster? Give reason for your opinion.

Exercise 2 (Synchronization) ([1, Exercise 2.1]) Consider the skeleton of the program in Listing 1 that prints all the lines in a file containing `pattern`.

1. Add the missing code for synchronizing access to `buffer`. Use the `await` statement for the synchronization code.
2. Extend your program so that it reads two files and prints all the lines that contain `pattern`. Identify the independent activities and use a separate process for each. Show all synchronization code that is required.

Exercise 3 (Producer-consumer) ([1, Exercise 2.2]) Consider the code of the simple producer-consumer problem in Listing 2. Change it so that the variable `p` is local to the producer process and `c` is local to the consumer process, not global. Hence, those variables cannot be used to synchronize access to `buf`.

Exercise 4 (Executions and atomicity) ([1, Exercise 2.10]) Consider the program in Listing 3.

1. Suppose each assignment statement is implemented by a single machine instruction and hence is atomic. How many possible executions are there? What are the possible final values of `x` and `y`?
2. Suppose each assignment statement is implemented by three atomic actions that load a register, add or subtract a value from that register, then store the result. How many possible executions are there now? What are the possible final values of `x` and `y`?

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

Listing 1: Finding patterns in a file (skeleton)

```
string buffer;   # contains one line of the input
bool done := false;
process Finder { # find patterns
    string line1;
    while (true) {
      wait for buffer to be full or done to be true;
      if (done) break;
      line1 := buffer;
      signal that buffer is empty;
      look for pattern in line1;
      if (pattern is in line1)
         write line1;
    }
}
process Reader { # read new lines
    string line2;
    while (true) {
      read next line of input into line2 or set EOF after last line;
      if (EOF) {done := true; break;}
      wait for buffer to be empty;
      buffer := line2;
      signal that buffer is full;
    }
}
```

Listing 2: Copying an array from a producer to a consumer; global `p` and `c`

```
int buffer, p := 0; c := 0;

process Producer {
  int a[N];
  while (p < N) {
    ⟨ await (p = c); ⟩
    buffer := a[p];
    p := p+1;
  }
}
process Consumer {
  int b[N];
  while (c < N) {
    ⟨ await (p > c); ⟩
    b[c] := buffer;
    c := c+1;
  }
}
```

Listing 3: A concurrent program with different executions

```
int x := 0, y := 0;
co
   x := x + 1; # S1
   x := x + 2; # S2
||
   x := x + 2; # P1
   y := y − x; # P2
oc
```

Exercise 5 (Interleaving, non-determinism, and atomicity) ([1, Exercise 2.12]) Consider the following program.

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

1. What are the possible final values of x and y?
2. Suppose the angle brackets are removed and each assignment statement is now implemented by three atomic actions: read the variables, add or multiply, and write to a variable. What are the possible final values of x and y now?

Exercise 6 (At most once) ([1, Exercise 2.14]) Consider the following program.

```
int x := 1, y := 1;
co
   <x := x + y;>      #S1
||
   y := 0;            #S2
||
   x := x − y;        #S3
oc
```

1. Do $S1, S2$ and $S3$ satisfy the requirements of the At-Most-Once Property?
2. What are the final values for x and y? Explain your answer.

Exercise 7 (AMO, termination) ([1, Exercise 2.15]) Consider the following program.

```
int x := 0, y := 10;

co
    while (x != y) x := x + 1;
||
    while (x != y) y := y − 1;
oc
```

1. Do all parts of the program meet the requirements of the At-Most-Once-Property?
2. Will the program terminate? Always? Sometimes? Never?

References

[1] G. R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming.* Addison-Wesley, 2000.