0.1 Synchronization

From WikipediA:
Synchronization is the coordination of events to operate a system in unison. The familiar conductor of an orchestra serves to keep the orchestra in time. Systems operating with all their parts in synchrony are said to be synchronous or in sync.

In Computer Science it is more complicated: there are many **synchronization constraints**. Examples:

- Serialization: Event A must happen before Event B.
- Mutual exclusion: Events A and B must not happen at the same time.

Clock idea: preset a time when the next event is allowed to starts. Probably will not work - often there is no clock to refer to, or multiple clocks – which are slightly off. Further problems: there is no guarantee how long a process or a thread will take to finish execution (think about context switching).

Solving above with messages:

```
You                          Bob
 a1:  Eat breakfast           b1:  Eat breakfast
 a2:  Work                    b2:  Wait for a call
 a3:  Eat lunch               b3:  Eat lunch
 a4:  Call Bob
```

$$a1 < a2 < a3 < a4$$
$$b1 < b2 < b3$$

where the relation $a1 < a2$ means that a1 happened before a2.

Message passing (call) guarantees $b2 > a4$. Which gives us:

$$b3 > b2 > a4 > a3$$

Definitions:

Events A and B are executed **sequentially** if there is a guarantee that A always happens before B (or B always before A). Otherwise we say that A and B are executed **concurrently**.

Note that concurrent does not mean simultaneously.

Non-determinism – the exact order of instructions is not defined.

print 1 print 2

Notes:

- Execution may be non-deterministic, but final result is deterministic. So such algorithm will still be called deterministic.
- How useful are non-deterministic algorithms? Consider algorithm for testing if number N is prime: 100 times choose a random number x from 1 to $\sqrt{N}$ and if x divides N print `not prime` otherwise print `possibly prime`.
- Another example: Monty Hall problem .....

0.2 Shared variables

Threads have local variables, local variable of one thread are not available to any other part of the program. Sometimes threads have to access same variable (object/data).

Thread A	Thread B
x = 5	x = 7
print x	print x

result is non-deterministic. What are the possible output/value pairs?

Race condition situation when result depends on the timing of 2 or more events.

Race conditions are very rarely part of the design, usually very hard to find bugs.

Race conditions may be hard to notice in your code:

Thread A	Thread B
count = count + 1	count = count + 1

`count = count + 1` is not **atomic** operation. It is Read, Increment, Write.

Definition: an operation that cannot be interrupted is said to be **atomic**.

Assume `count` is a shared variable:

Thread A	Thread B
a1: t = count	b1: t = count
a2: ++t;	b2: ++t;
a3: count = t	b3: count = t

Now – order a1-a2-a3-b1-b2-b3 produces 2, while a1-b1-a2-b2-a3-b3 will produce 1.

What about this?

Thread A	Thread B
++count;	++count;

Is increment of integer (`int`) an atomic operation?