# Race condition

A race condition is a situation in which the result of executing two or more processes in parallel can depend on the relative timing of the execution of the processes. In some situations, this may be expected. In other situations, it is unexpected and a bug in the program.

Here's an example. Consider two processes P1 and P2 that increment a single "shared variable" x at the "same time".

| Process P1 | Process P2 |
|---|---|
| … | … |
| x= x + 1; | x= x + 1; |

Suppose x is initially 2. Then after both processes increment x, x should contain 4.

But execution of an assignment statement is not an "atomic action"; it is performed not in one step but in three: load x into a "register", add 1 to the register, and store the register in x. Below, we see how interleaving of these three steps by P1 and P2 could end up with 3, and not 4, in x.

| | **Process P1** | **Process P2** |
|---|---|---|
| (1) | Load x into a register A | |
| (2) | Add 1 to register A | |
| (3) | | Load x in a register B |
| (4) | | Add 1 to register B |
| (5) | Store register A in x | |
| (6) | | Store register B in x |

If a race condition can occur at such a low level, with an assignment to a simple variable, you can imagine that it can occur with higher-level shared data structures —with a stack, a tree, a graph, or a database. Parallel execution, or concurrency, introduces problems that are much harder to solve than with a sequential process acting only on its own variables.