TDT4200 Problem Set 4

What is a critical section?

Critical sections are specific code sections where execution may not overlap. Ignoring this requirement leads to undefined behaviour.

How can a critical section be protected from race conditions?

There some options available. Here are just the ones mentioned in the lecture slides:

- Atomic operations; These cannot overlap
- Load Linked & Store Conditional
- Mutexes and semaphores
- Constructs from higher-level languages

What is oversubscription?

Oversubscription is simply creating more threads than the amount of available compute units. In some cases this allows compute units to be busy for longer as oversubscribed threads can use CPU time when another thread is waiting.

Problem:

Suppose that you have a set of n pthreads, each with a local integer that represents each thread's individual index from 0 through n − 1, and a print statement at the end of its function body. How would you implement a method to ensure that the output of the print statements appears in the order of the thread indices? A pseudo-code description is sufficient to answer this question.

```
// We assume that we have the thread's index
let mut thread: u32 = ▉;
// Thread zero does not need to wait
if thread > 0 {
    join(thread - 1); // Wait for previous thread
}
println!("Thread {} here!", thread);
return 0;
```

Of course, this could also be extracted into a seperate function with the thread index as an argument.

Stats & Speedups

All tests ran on an i7-8705G (8 threads)

Program	Time	Comment
Sequential	73.9 s	Slow, as expected. Easy to debug though!
Pthreads (4 threads)	39.7 s	~1.8x improvement. Not as good as I hoped, likely because I used a lot of barriers in the main loop.
Pthreads (8 threads)	38.8 s	No improvement, probably because the OS is busy with other stuff
OMP	108.4 s	This was *very* surprising but ultimately to be expected. There's no way that the OS will just let our application hog the entire CPU. While waiting I observed that the programm actually used about 6 threads on average. This creates a *lot* of idle waiting due to barriers and such.
OMP (4 threads override)	34.9 s	Reducing the thread count to 4 gives a nice performance boost!