## Week 2

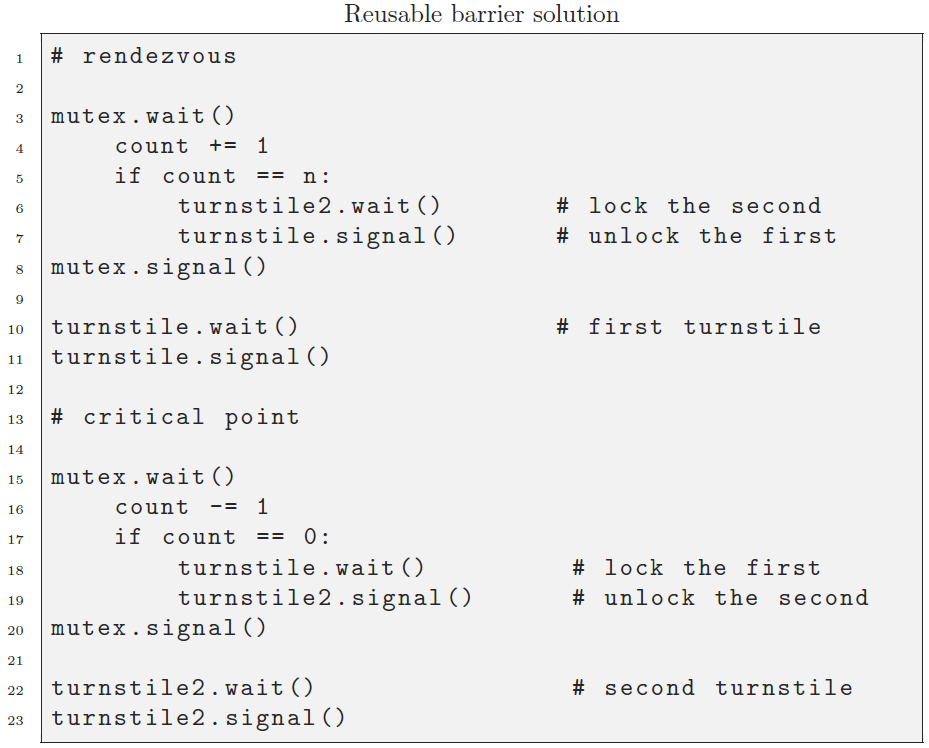
### E 3.7 Reusable Barrier (I)

Implement the Reusable Barrier of paragraph 3.7, but only with the use of semaphores (so no counters). The number of threads is known at compile time, e.g. 4.

### F 3.7 Reusable Barrier (II)

Re-implement the solution of the Reusable Barrier of paragraph 3.7, but don't use turnstile.wait() for locking (aka closing) a turnstile (see the rectangles in the following picture).

Tip: do not start with the code as given in LBoS and move some statements around until it more or less seems to work, but start with an empty sheet and write a clean implementation.



## Answer

For these 2 assignments we designed 2 different solutions.

Our first solution was done using asymmetry, and involved n+1 threads. The extra threads is in charge of scheduling when the other n threads are allowed to proceed. The code for this solution can be found in the file: assignment\_EF\_assymetric.md

Our second solution has several implementations. One using semaphores and variables, and one using just variables. The underlying mechanism for both is the same, in that they involve calling wait() on a specific semaphore numerous amount of times until the right amount of signals have been given by other waiting threads. These 2 implementations can be found in the file: assignment\_EF\_symmetric.md

Note: the bottom of the file contains the solution that was provided by school.

### G 3.8 Queue: followers & leaders

Make a symmetric implementation of the 3.8 problem with a pipet; without counters.

Ensure that an arbitrary number of follower and leader threads can be started (e.g. N=5)

## Answer

For this we created a solution in which the pipet acts as both the mutex as well as the queue. Inside of this pipet we use a rendezvous to align the 2 threads with one another. The code for this can be found in the file: assignment\_G.md